

Uncertain Spatiotemporal Logic for General Intelligence

Nil Geisweiller and Ben Goertzel

Novamente LLC

Abstract

Spatiotemporal reasoning is an important skill that an AGI is expected to have, innately or not. Much work has already been done in defining reasoning systems for space, time and spacetime, such as the Region Connection Calculus for space, Allen’s Interval Algebra for time, or the Qualitative Trajectory Calculus for motion. However, these reasoning systems rarely take adequate account of uncertainty, which poses an obstacle to using them in an AGI system confronted with an uncertain reality. In this paper we show how to use PLN (Probabilistic Logic Networks) to represent spatiotemporal knowledge and reasoning, via incorporating existing spatiotemporal calculi, and considering a novel extension of standard PLN truth values inspired by $\mathcal{P}(\mathcal{Z})$ -logic. This “PLN-ization” of existing spatiotemporal calculi, we suggest, constitutes an approach to spatiotemporal inference suitable for use in AGI systems that incorporate logic-based components.

Introduction

Most of the problems and situations humans confront every day involve space and time explicitly and centrally. Thus, any AGI system aspiring to even vaguely reach humanlike intelligence must have some reasonably efficient and general means to solve spatiotemporal problems. Multiple alternate or complementary methodologies may be used to achieve this, including spatiotemporal logical inference, internal simulation, or techniques like recurrent neural nets whose dynamics defy simple analytic explanation. We focus here on spatiotemporal logical inference, addressing the problem of creating a spatiotemporal logic adequate for use within an AGI system that confronts the same sort of real-world problems that humans typically do.

Should Spatiotemporal Intuition Be Preprogrammed Or Learned? In principle, one might argue, an AGI should be able to *learn* to reason about space and time just like anything else, obviating the need for spatiotemporal logic or other pre-programmed mechanisms. This would clearly be true of a highly powerful AGI system like (the purely theoretical) AIXI^{tl}. However this kind of foundational learning about space and time may be objectionably costly in practice. Also, it seems clear that some fundamental intuition for space and time is hard-coded into the human infant’s brain

(Joh05), which provides conceptual motivation for supplying AGI systems with some a priori spatiotemporal knowledge.

Overview A great deal of excellent work has already been done in the areas of spatial, temporal and spatiotemporal reasoning, such as the Region Connection Calculus (RCC93) for topology, the Cardinal Direction Calculus (LLR09) for direction, Allen’s Interval Algebra for time, or the Qualitative Trajectory Calculus for motion. Extensions to deal with uncertainty have been introduced too. However, we believe, they do not quite provide an adequate foundation for a logic-incorporating AGI system to do spatiotemporal reasoning. For instance, according to a fuzzy extension of RCC as developed in (SDCCCK08), asking how much Z is a part of X knowing how much Y is a part of X and Z is a part of Y (see Figure 1) would result in the answer $[0, 1]$ (a state of total ignorance), as Z can be either totally part of X or not at all. For that reason we consider probability distributions of fuzzy values (Yan09) rather than fuzzy values or intervals of fuzzy values.

So we will show how to represent spatiotemporal knowledge via incorporating existing spatiotemporal calculi into the PLN (GIGH08) uncertain reasoning framework, and then show how to carry out spatiotemporal logical inference using PLN inference rules.

Uncertainty with Distributional Fuzzy Values

The uncertainty extension we use is inspired by $\mathcal{P}(\mathcal{Z})$ (Yan09), an extension of fuzzy logic that considers distributions of fuzzy values rather than mere fuzzy values. For instance the connector \neg (often defined as $\neg x = 1 - x$) is extended into a connector such that the resulting density function is $\mu_{\neg}(x) = \mu(1 - x)$ where μ is the probability density function of the argument.

We define a wider class of connectors that can modulate the output of the distribution. Let $F : [0, 1]^n \mapsto ([0, 1] \mapsto \mathbb{R}^+)$ be a n -ary connector that takes n fuzzy values and returns a probability density function. In that case the probability density function $\mu_F : [0, 1] \mapsto \mathbb{R}^+$ resulting from the extension of F over density functions is:

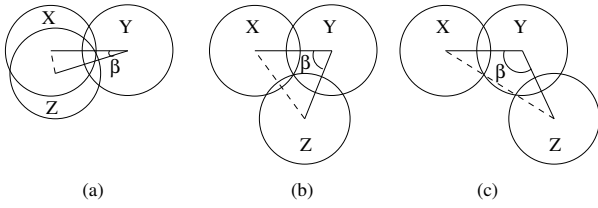


Figure 1: d_{XZ} , in dashline, for 3 different angles

$$\mu_F = \underbrace{\int_0^1 \dots \int_0^1}_n F(x_1, \dots, x_n) \mu_1(x_1) \dots \mu_n(x_n) dx_1 \dots dx_n \quad (1)$$

where μ_1, \dots, μ_n are the n input arguments. Let us give an example of such a connector with a fuzzy version of the RCC relationship PartOf (P for short). A typical inference rule in the crisp case would be:

$$\frac{P(X, Y) \quad P(Y, Z)}{P(X, Z)} \quad (2)$$

expressing the transitivity of P. But using a distribution of fuzzy values we would have the following rule

$$\frac{P(X, Y) \langle \mu_1 \rangle \quad P(Y, Z) \langle \mu_2 \rangle}{P(X, Z) \langle \mu_{POT} \rangle} \quad (3)$$

POT stands for PartOf Transitivity. The definition of μ_{POT} for that particular inference rule may depend on many assumptions like the shapes and sizes of regions X , Y and Z . We have worked out the exact definition of μ_{POT} based on simplified assumptions (regions are unitary circles) in the extended version of this paper.

It should not be too hard to derive a more realistic formula based on other more complex assumptions. Though another possibility would be to let the system learn POT (as well as other connectors) based on its experience. Because it is not obvious what are the right assumptions in the first place. So the agent would initially perform spatial reasoning not too accurately, but would improve over time.

Of course the rule could also be extended to involve more premises containing information about sizes and shapes of the regions.

Simplifying Numerical Calculation Using probability density as described above is computationally expensive. To decrease computational cost, several cruder approaches are possible, such as discretizing the probability density functions with a coarse resolution, or restricting attention to beta distributions and treating only their means and variances (as in (Yan09)).

Example of Spatio-temporal Inference in PLN

We now give an example of spatiotemporal inference rules coded in PLN. This paper is too short to contain examples

of real-world commonsense inferences, but we invite the author to visit the OpenCog project Wiki web page which contains a few examples¹.

Although the current implementation of PLN incorporates both fuzziness and probability it does not have a built-in truth value to represent distributional fuzzy values. However, we intend to add that extension to the PLN implementation in the near future, and for our present theoretical purposes we will just assume that such a distributional fuzzy value exists, let us call it *DF Truth Value*.

Here is an example of the inference rule expressing the transitivity for the relationship PartOf

```

ForAllLink $X $Y $Z
  ImplicationLink
    ANDLink
      PartOf($X, $Y) <tv1>
      PartOf($Y, $Z) <tv2>
    ANDLink
      tv3 =  $\mu_{POT}(tv_1, tv_2)$ 
      PartOf($X, $Z) <tv3>

```

(4)

Conclusion

Every AGI system that aspires to humanlike intelligence must carry out spatiotemporal inference in some way. Logic is not the only way to carry out spatiotemporal inference broadly construed. But if one is going to use logic, we believe the most effective approach is to incorporate specific spatiotemporal calculi, extended to encompass distributional fuzzy truth values. The next step is to implement it in the OpenCog implementation of PLN, and carry out a large number of practical examples. Alongside their direct practical value, these examples will teach us a great deal about uncertain spatiotemporal logic, including issues such as the proper settings of the various parameters and the customization of inference control mechanisms.

References

- [GIGH08] Ben Goertzel, Matthew Ikl, Izabela Freire Goertzel, and Ari Heljakka. *Probabilistic Logic Networks: A Comprehensive Framework for Uncertain Inference*. Springer Publishing Company, Incorporated, 2008.
- [Joh05] Mark Johnson. *Developmental Cognitive Neuroscience*. Wiley-Blackwell, 2005.
- [LLR09] Weiming Liu, Sanjiang Li, and Jochen Renz. Combining rcc-8 with qualitative direction calculi: Algorithms and complexity. In *IJCAI*, 2009.
- [RCC93] D. A. Randell, Z. Cui, and A. G. Cohn. A spatial logic based on regions and connection. 1993.
- [SDCCK08] Steven Schockaert, Martine De Cock, Chris Cornelis, and Etienne E. Kerre. Fuzzy region connection calculus: An interpretation based on closeness. *Int. J. Approx. Reasoning*, 48(1):332–347, 2008.
- [Yan09] King-Yin Yan. Genifer an artificial general intelligence. 2009. <https://launchpad.net/agi-book>.

¹<http://www.opencog.org/wiki/Spatiotemporal.Inference>

Compression-Driven Progress in Science

Leo Pape

Utrecht University
Utrecht, The Netherlands
l.pape@geo.uu.nl

Abstract

The construction of an *artificial scientist*, a machine that discovers and describes the general rules governing a variety of complex environments, can be considered an important challenge for artificial general intelligence. Recently, a computational framework for scientific investigation has been postulated in the theory of compression-driven progress. Here, I propose an implementation of an artificial scientist based on the compression principle, and explore the possibilities and challenges for its application in scientific research.

Introduction

Human beings reckon scientific understanding of the world among the most powerful of their abilities (e.g. Rorty, 1991), and it is therefore not surprising that researchers try to simulate this capability with computer programs and robots (e.g. King et al., 2009; Schmidt and Lipson, 2009). Such machines, called *artificial scientists*, not only enhance our ability to carry out scientific research, but building them also guides our understanding how human scientists come to comprehend the world. Creating an artificial scientist that is not restricted to a specific domain, but performs scientific research *in general* can be considered a great challenge for artificial general intelligence.

To construct an artificial scientist, we need to have some idea of *what* it is that human scientists do and *how* they do this. Since the societal, fundamental or personal goals of science are not contained in its domain, I here rather define the *activity* of scientists as the development of theories that explain the past and predict the future, and are consistent with other theories. These theories result from systematic reasoning about observations, whether obtained accidentally or intentionally, for example in a controlled experiment.

A theory that not only explains how scientific progress is achieved by human beings, but also specifies how scientific investigation can be carried out with computer algorithms, is the theory of compression-driven progress (Schmidhuber, 2009). This theory considers both human and artificial scientists as computationally limited observers that try to represent observations in an efficient manner. Finding efficient representations

entails identifying regularities that allow the observer to compress the original observations and predict future observations. Discovered regularities then serve as an explanation for the observed phenomena. Compression *progress* is achieved when an observer discovers previously unknown regularities that provide increased compression of observations. The theory of compression-driven progress further postulates that scientists direct their attention to interesting data, that is, data that is neither impossible to compress (i.e. truly random) nor easily compressible with existing methods, but is expected to hold previously unknown regularities that allow for further compression.

Based on this theory, it is possible to implement an artificial scientist that can operate in a variety of scientific disciplines. In this paper I explore the possibilities and challenges for the construction of a compression-driven artificial scientist.

Compression-Driven Artificial Scientists

A compression-driven artificial scientist is a machine that aims to predict future and unobserved observations by identifying the regularities underlying its sensory input. It consists of the following components: (1) A sensory input module that collects observations, such as a camera, microphone or software interface. (2) An adaptive compressor that discovers regularities in the observational data. A compressor that is particularly suitable for this task is the deep autoencoder of Hinton and Salakhutdinov (2006), which learns to convert high-dimensional input data to short codes. Of course it is possible to use another, possibly even more general algorithm, but the Hinton and Salakhutdinov autoencoder has the advantage that it can *reconstruct* and thus *predict* data from its coded representations. (3) A reinforcement learning algorithm that learns to select actions (e.g. manipulate the world, perform experiments, direct attention, move) that take the artificial scientist to interesting data. *Interestingness* is defined as the improvement of the adaptive compressor on parts of the input data, and is determined from the number of bits needed to reconstruct the original input from its coded representation. (4) Optionally, a physical implementation, such as a robot. The use of existing datasets,

however, allows for the implementation of the artificial scientist as a software program, which can significantly reduce the costs and complexity of its construction.

Representation

The compression-driven artificial scientist is not immediately useful to its human colleagues, because the regularities it discovers are not represented in understandable form (i.e. in a connectionist architecture). A related problem is that the artificial scientist has no a-priori notion of objects¹ in its raw sensory inputs (e.g. a stream of bits or numbers), while its theories should preferably be about such objects, not about bits or numbers. These two problems reflect the more general challenge of constructing symbolic representations from subsymbolic data (see e.g. Smolensky, 1988). Here I explain how both artificial and human scientists construct mental *objects* from sensory inputs using the compression principle, and how this process is the basis for communicating discovered regularities in symbolic form.

Using the basic operations of its reasoning apparatus, the artificial scientist builds methods that compress those parts of its sensory input signal that have certain structure. Note that compression does not merely apply to static objects in space, but also extends to structural relations in time. Different types of structure require different compression methods, allowing the artificial scientist to distinguish individual entities or phenomena by their *method of compression*. When compression methods are organized in a hierarchical fashion, the artificial scientist can construct more abstract concepts and find increasingly general relations between objects on different abstraction levels.

Human scientists discovered many parts of the world that are compressible and predictable to some extent, while other parts seem to resist compression and prediction. Interestingly, the inability to describe and predict certain parts of the world is mostly not because the fundamental forces of nature are unknown to science, but because the deterministic laws of nature produce chaos in some parts and order in other parts of the world (where chaos and order are equivalent to incompressible and compressible observations, respectively). That is, the most fundamental relations express only the most general aspects of the world, not all specific details relevant to our lives. Human scientists therefore try to find intermediate levels on which the world exhibits regularity, give those parts names and relate them in a systematic way to already identified entities. As a result, different *levels of organization* materialize into individual objects¹ of scientific thought.

Discovered structure in parts of the world can only be communicated in a meaningful sense through a shared language. While mathematics and logic are rather pop-

ular languages in science, the relations they express have no intrinsic meaning, but need to be related to concepts that are recognized by all communicating parties (e.g. Schmidt and Lipson (2009) used symbolic regression on variables whose human interpretation was established beforehand, not discovered independently by their algorithms). Artificial scientists therefore need to learn how to map their internal representations of discovered objects and structure onto the entities (e.g. symbols) of a shared language. Such a shared language can, in principle, be learned among different instances of artificial scientists in an *unsupervised* fashion. However, this artificial language is probably not easily accessible to human scientists. Instead, an artificial scientist should learn a language that is easily understandable for human scientists. For this, the artificial scientist needs to learn from labeled data, either by augmenting the reinforcement learning algorithm with an external reward based on label prediction, or by a function (e.g. an additional neural network) that learns to map internal representations onto labels in a supervised fashion.

Conclusion

In this paper I explored the possibilities and challenges for the construction of a compression-driven artificial scientist. While the theory of compression-driven progress provides the basic mechanism for scientific investigation, an ongoing challenge is the human interpretation of theories constructed by artificial scientists. In the future I aim to implement the proposed architecture and demonstrate its capability to discover known and novel forms of structure in scientific data.

References

- Hinton, G. E., and Salakhutdinov, R. R. 2006. Reducing the dimensionality of data with neural networks. *Science* 313(5786):504–507.
- King, R. D.; Rowland, J.; Oliver, S. G.; Young, M.; Aubrey, W.; Byrne, E.; Liakata, M.; Markham, M.; Pir, P.; Soldatova, L. N.; Sparkes, A.; Whelan, K. E.; and Clare, A. 2009. The automation of science. *Science* 324:85–89.
- Rorty, R. 1991. *Objectivity, relativism, and truth: philosophical papers*, volume 1. Cambridge, UK: Cambridge University Press.
- Schmidhuber, J. 2009. *Driven by compression progress: A simple principle explains essential aspects of subjective beauty, novelty, surprise, interestingness, attention, curiosity, creativity, art, science, music, jokes*. Lecture Notes in Computer Science. Berlin / Heidelberg: Springer. 48–76.
- Schmidt, M., and Lipson, H. 2009. Distilling free-form natural laws from experimental data. *Science* 324:81–85.
- Smolensky, P. 1988. A proper treatment of connectionism. *Behavioural and Brain Sciences* 11:1–74.

¹objects in the most general sense, such as material objects like molecules and robots, but also more abstract objects like a rainbow, a supercluster (of galaxies) or musical notes

Toward a Formal Characterization of Real-World General Intelligence

Ben Goertzel

Novamente LLC
1405 Bernerd Place
Rockville MD 20851

Abstract

Two new formal definitions of intelligence are presented, the "pragmatic general intelligence" and "efficient pragmatic general intelligence." Largely inspired by Legg and Hutter's formal definition of "universal intelligence," the goal of these definitions is to capture a notion of general intelligence that more closely models that possessed by humans and practical AI systems, which combine an element of universality with a certain degree of specialization to particular environments and goals. Pragmatic general intelligence measures the capability of an agent to achieve goals in environments, relative to prior distributions over goal and environment space. Efficient pragmatic general intelligence measures this same capability, but normalized by the amount of computational resources utilized in the course of the goal-achievement. A methodology is described for estimating these theoretical quantities based on observations of a real biological or artificial system operating in a real environment. Finally, a measure of the "degree of generality" of an intelligent system is presented, allowing a rigorous distinction between "general AI" and "narrow AI."

Introduction

"Intelligence" is a commonsense, "folk psychology" concept, with all the imprecision and contextuality that this entails. One cannot expect any compact, elegant formalism to capture all of its meanings. Even in the psychology and AI research communities, divergent definitions abound; Legg and Hutter (LH07a) lists and organizes 70+ definitions from the literature.

Practical study of natural intelligence in humans and other organisms, and practical design, creation and instruction of artificial intelligences, can proceed perfectly well without an agreed-upon formalization of the "intelligence" concept. Some researchers may conceive their own formalisms to guide their own work, others may feel no need for any such thing.

But nevertheless, it is of interest to seek formalizations of the concept of intelligence, which capture useful fragments of the commonsense notion of intelligence, and provide guidance for practical research in cognitive science and AI. A number of such formalizations have been given in recent decades, with varying degrees

of mathematical rigor. Perhaps the most carefully-wrought formalization of intelligence so far is the theory of "universal intelligence" presented by Shane Legg and Marcus Hutter in (LH07b), which draws on ideas from algorithmic information theory.

Universal intelligence captures a certain aspect of the "intelligence" concept very well, and has the advantage of connecting closely with ideas in learning theory, decision theory and computation theory. However, the kind of general intelligence it captures best, is a kind which is in a sense *more general* in scope than human-style general intelligence. Universal intelligence does capture the sense in which humans are more intelligent than worms, which are more intelligent than rocks; and the sense in which theoretical AGI systems like Hutter's AIXI or $AIXI^{tl}$ (Hut05) would be much more intelligent than humans. But it misses essential aspects of the intelligence concept as it is used in the context of intelligent natural systems like humans or real-world AI systems.

Our main goal here is to present variants of universal intelligence that better capture the notion of intelligence as it is typically understood in the context of real-world natural and artificial systems. The first variant we describe is *pragmatic general intelligence*, which is inspired by the intuitive notion of intelligence as "the ability to achieve complex goals in complex environments," given in (Goe93). After assuming a prior distribution over the space of possible environments, and one over the space of possible goals, one then defines the pragmatic general intelligence as the expected level of goal-achievement of a system relative to these distributions. Rather than measuring truly broad mathematical general intelligence, pragmatic general intelligence measures intelligence in a way that's specifically biased toward certain environments and goals.

Another variant definition is then presented, the *efficient pragmatic general intelligence*, which takes into account the amount of computational resources utilized by the system in achieving its intelligence. Some argue that making efficient use of available resources is a defining characteristic of intelligence, see e.g. (Wan06).

A critical question left open is the characterization of the prior distributions corresponding to everyday hu-

man reality; we have given a semi-formal sketch of some ideas on this in a prior conference paper (Goe09), where we present the notion of a "communication prior," which assigns a probability weight to a situation S based on the ease with which one agent in a society can communicate S to another agent in that society, using multimodal communication (including verbalization, demonstration, dramatic and pictorial depiction, etc.). We plan to develop this and related notions further.

Finally, we present a formal measure of the "generality" of an intelligence, which precisiates the informal distinction between "general AI" and "narrow AI."

Legg and Hutter's Definition of General Intelligence

First we review the definition of general intelligence given in (LH07b), as the formal setting they provide will also serve as the basis for our work here.

We consider a class of active agents which observe and explore their environment and also take actions in it, which may affect the environment. Formally, the agent sends information to the environment by sending symbols from some finite alphabet called the *action space* Σ ; and the environment sends signals to the agent with symbols from an alphabet called the *perception space*, denoted \mathcal{P} . Agents can also experience rewards, which lie in the *reward space*, denoted \mathcal{R} , which for each agent is a subset of the rational unit interval.

The agent and environment are understood to take turns sending signals back and forth, yielding a history of actions, observations and rewards, which may be denoted

$$a_1 o_1 r_1 a_2 o_2 r_2 \dots$$

or else

$$a_1 x_1 a_2 x_2 \dots$$

if x is introduced as a single symbol to denote both an observation and a reward. The complete interaction history up to and including cycle t is denoted $ax_{1:t}$; and the history before cycle t is denoted $ax_{<t} = ax_{1:t-1}$.

The agent is represented as a function $\pi =$ which takes the current history as input, and produces an action as output. Agents need not be deterministic, an agent may for instance induce a probability distribution over the space of possible actions, conditioned on the current history. In this case we may characterize the agent by a probability distribution $\pi(a_t | ax_{<t})$. Similarly, the environment may be characterized by a probability distribution $\mu(x_k | ax_{<k} a_k)$. Taken together, the distributions π and μ define a probability measure over the space of interaction sequences.

To define universal intelligence, Legg and Hutter consider the class of environments that are *reward-summable*, meaning that the total amount of reward they return to any agent is bounded by 1. Where r_i denotes the reward experienced by the agent from the

environment at time i , the *expected total reward* for the agent π from the environment μ is defined as

$$V_\mu^\pi \equiv E\left(\sum_{i=1}^{\infty} r_i\right) \leq 1$$

To extend their definition in the direction of greater realism, we first introduce a second-order probability distribution ν , which is a probability distribution over the space of environments μ . The distribution ν assigns each environment a probability. One such distribution ν is the Solomonoff-Levin universal distribution in which one sets $\nu = 2^{-K(\mu)}$; but this is not the only distribution ν of interest. In fact a great deal of real-world general intelligence consists of the adaptation of intelligent systems to particular distributions ν over environment-space, differing from the universal distribution. We then define

Definition 1. *The biased universal intelligence of an agent π is its expected performance with respect to the distribution ν over the space of all computable reward-summable environments, E , that is,*

$$\Upsilon(\pi) \equiv \sum_{\mu \in E} \nu(\mu) V_\mu^\pi$$

Legg and Hutter's **universal intelligence** is obtained by setting ν equal to the universal distribution.

This framework is more flexible than it might seem. E.g. suppose one wants to incorporate agents that die. Then one may create a special action, say a_{666} , corresponding to the state of death, to create agents that

- in certain circumstances output action a_{666}
- have the property that if their previous action was a_{666} , then all of their subsequent actions must be a_{666}

and to define a reward structure so that actions a_{666} always bring zero reward. It then follows that death is generally a bad thing if one wants to maximize intelligence. Agents that die will not get rewarded after they're dead; and agents that live only 70 years, say, will be restricted from getting rewards involving long-term patterns and will hence have specific limits on their intelligence.

Connecting Legg and Hutter's Model of Intelligent Agents to the Real World

A notable aspect of the Legg and Hutter formalism is the separation of the reward mechanism from the cognitive mechanisms of the agent. While commonplace in the reinforcement learning literature, this seems psychologically unrealistic in the context of biological intelligences and many types of machine intelligences. Not all human intelligent activity is specifically reward-seeking in nature; and even when it is, humans often pursue complexly constructed rewards, that are defined in terms of their own cognitions rather than separately given. Suppose a certain human's goals are true love, or world peace,

and the proving of interesting theorems – then these goals are defined by the human herself, and only she knows if she’s achieved them. An externally-provided reward signal doesn’t capture the nature of this kind of goal-seeking behavior, which characterizes much human goal-seeking activity (and will presumably characterize much of the goal-seeking activity of advanced engineered intelligences also) ... let alone human behavior that is spontaneous and unrelated to explicit goals, yet may still appear commonsensically intelligent.

One could seek to bypass this complaint about the reward mechanisms via a sort of “neo-Freudian” argument, via

- associating the reward signal, not with the “external environment” as typically conceived, but rather with a portion of the intelligent agent’s brain that is separate from the cognitive component
- viewing complex goals like true love, world peace and proving interesting theorems as indirect ways of achieving the agent’s “basic goals”, created within the agent’s memory via subgoal mechanisms

but it seems to us that a general formalization of intelligence should not rely on such strong assumptions about agents’ cognitive architectures. So below, after introducing the pragmatic and efficient pragmatic general intelligence measures, we will propose an alternate interpretation wherein the mechanism of external rewards is viewed as a theoretical test framework for assessing agent intelligence, rather than a hypothesis about intelligent agent architecture.

In this alternate interpretation, formal measures like the universal, pragmatic and efficient pragmatic general intelligence are viewed as *not* being directly applicable to real-world intelligences, because they involve the behaviors of agents over a wide variety of goals and environments, whereas in real life the opportunity to observe an agent’s activities are much more limited. However, they are viewed as being *indirectly* applicable to real-world agents, in the sense that an external intelligence can observe an agent’s real-world behavior and then *infer* its likely intelligence according to these measures.

In a sense, this interpretation makes our formalized measures of intelligence the opposite of real-world IQ tests. An IQ test is a quantified, formalized test which is designed to approximately predict the informal, qualitative achievement of humans in real life. On the other hand, the formal definitions of intelligence we present here are quantified, formalized tests that are designed to capture abstract notions of intelligence, but which can be approximately evaluated on a real-world intelligent system by observing what it does in real life.

Pragmatic General Intelligence

To formalize pragmatic general intelligence, the first modification we need to introduce to Legg and Hutter’s framework is to allow agents to maintain memories (of

finite size), and at each time step to carry out internal actions on their memories as well as external actions in the environment. Legg and Hutter, in their theory of universal intelligence, don’t need to worry about memory, because their definition of intelligence doesn’t take into account the computational resource usage of agents. Thus, in their framework, it’s acceptable for an agent to determine its actions based on the entire past history of perceptions, actions and rewards. On the other hand, if an agent needs to conserve memory and/or memory access time, it may not be practical for it to store its entire history, so it may need to store a sample thereof, and/or a set of memory items representing useful abstractions of its history. If one is gauging intelligence using a measure that incorporates space and time resource utilization, then the size and organization of this memory become important aspects of the system’s intelligence.

Further extending the Legg and Hutter framework, we introduce the notion of a *goal-seeking agent*. We define goals as mathematical functions (to be specified below) associated with symbols drawn from the alphabet \mathcal{G} ; and we consider the environment as sending goal-symbols to the agent along with regular observation-symbols. (Note however that the presentation of a goal-symbol to an agent does not necessarily entail the explicit communication to the agent of the contents of the goal function. This must be provided by other, correlated observations.) We also introduce a conditional distribution $\gamma(g, \mu)$ that gives the weight of a goal g in the context of a particular environment μ .

In this extended framework, an interaction sequence looks like

$$m_1 a_1 o_1 g_1 r_1 m_2 a_2 o_2 g_2 r_2 \dots$$

or else

$$w_1 y_1 w_2 y_2 \dots$$

if w is introduced as a single symbol to denote the combination of a memory action and an external action, and y is introduced as a single symbol to denote the combination of an observation, a reward and a goal.

Each goal function maps each finite interaction sequence $I_{g,s,t} = ay_{s:t}$ with g_s corresponding to g , into a value $r_g(I_{g,s,t}) \in [0, 1]$ indicating the value or “raw reward” of achieving the goal during that interaction sequence. The total reward r_t obtained by the agent is the sum of the raw rewards obtained at time t from all goals whose symbols occur in the agent’s history before t . We will use “context” to denote the combination of an environment, a goal function and a reward function.

If the agent is acting in environment μ , and is provided with g_s corresponding to g at the start of the time-interval $T = \{i \in (s, \dots, t)\}$, then the *expected goal-achievement* of the agent, relative to g , during the interval is the expectation

$$V_{\mu,g,T}^{\pi} \equiv E\left(\sum_{i=s}^t r_g(I_{g,s,i})\right)$$

where the expectation is taken over all interaction sequences $I_{g,s,i}$ drawn according to μ . We then propose

Definition 2. *The pragmatic general intelligence of an agent π , relative to the distribution ν over environments and the distribution γ over goals, is its expected performance with respect to goals drawn from γ in environments drawn from ν ; that is,*

$$\Pi(\pi) \equiv \sum_{\mu \in E, g \in \mathcal{G}, T} \nu(\mu) \gamma(g, \mu) V_{\mu,g,T}^{\pi}$$

(in those cases where this sum is convergent).

This definition formally captures the notion that "intelligence is achieving complex goals in complex environments," where "complexity" is gauged by the assumed measures ν and γ .

If ν is taken to be the universal distribution, and γ is defined to weight goals according to the universal distribution, then pragmatic general intelligence reduces to universal intelligence.

Furthermore, it is clear that a universal algorithmic agent like AIXI (Hut05) would also have a high pragmatic general intelligence, under fairly broad conditions. As the interaction history grows longer, the pragmatic general intelligence of AIXI would approach the theoretical maximum; as AIXI would implicitly infer the relevant distributions via experience. However, if significant reward discounting is involved, so that near-term rewards are weighted much higher than long-term rewards, then AIXI might compare very unfavorably in pragmatic general intelligence, to other agents designed with prior knowledge of ν and γ in mind.

The most interesting case to consider is where ν and γ are taken to embody some particular bias in a real-world space of environments and goals, and this biases is appropriately reflected in the internal structure of an intelligent agent. Note that an agent need not lack universal intelligence in order to possess pragmatic general intelligence with respect to some non-universal distribution over goals and environments. However, in general, given limited resources, there may be a tradeoff between universal intelligence and pragmatic intelligence. Which leads to the next point: how to encompass resource limitations into the definition.

One might argue that the definition of Pragmatic General Intelligence is already encompassed by Legg and Hutter's definition because one may bias the distribution of environments within the latter by considering different Turing machines underlying the Kolmogorov complexity. However this is not a general equivalence because the Solomonoff-Levin measure intrinsically decays exponentially, whereas an assumptive distribution over environments might decay at some other rate. This issue seems to merit further mathematical investigation.

Incorporating Computational Cost

Let $\eta_{\pi,\mu,g,T}$ be a probability distribution describing the amount of computational resources consumed by an agent π while achieving goal g over time-scale T . This is a probability distribution because we want to account for the possibility of nondeterministic agents. So, $\eta_{\pi,\mu,g,T}(Q)$ tells the probability that Q units of resources are consumed. For simplicity we amalgamate space and time resources, energetic resources, etc. into a single number Q , which is assumed to live in some subset of the positive reals. Space resources of course have to do with the size of the system's memory, briefly discussed above. Then we may define

Definition 3. *The efficient pragmatic general intelligence of an agent π with resource consumption $\eta_{\pi,\mu,g,T}$, relative to the distribution ν over environments and the distribution γ over goals, is its expected performance with respect to goals drawn from γ in environments drawn from ν , normalized by the amount of computational effort expended to achieve each goal; that is,*

$$\Pi_{Eff}(\pi) \equiv \sum_{\mu \in E, g \in \mathcal{G}, Q, T} \frac{\nu(\mu) \gamma(g, \mu) \eta_{\pi,\mu,g,T}(Q)}{Q} V_{\mu,g,T}^{\pi}$$

(in those cases where this sum is convergent).

Efficient pragmatic general intelligence is a measure that rates an agent's intelligence higher if it uses fewer computational resources to do its business.

Note that, by abandoning the universal prior, we have also abandoned the proof of convergence that comes with it. In general the sums in the above definitions need not converge; and exploration of the conditions under which they do converge is a complex matter.

Assessing the Intelligence of Real-World Agents

The pragmatic and efficient pragmatic general intelligence measures are more "realistic" than the Legg and Hutter universal intelligence measure, in that they take into account the innate biasing and computational resource restrictions that characterize real-world intelligence. But as discussed earlier, they still live in "fantasy-land" to an extent – they gauge the intelligence of an agent via a weighted average over a wide variety of goals and environments; and they presume a simplistic relationship between agents and rewards that does not reflect the complexities of real-world cognitive architectures. It is not obvious from the foregoing how to apply these measures to real-world intelligent systems, which lack the ability to exist in such a wide variety of environments within their often brief lifespans, and mostly go about their lives doing things other than pursuing quantified external rewards. In this brief section we describe an approach to bridging this gap. The treatment is left-semi-formal in places.

We suggest to view the definitions of pragmatic and efficient pragmatic general intelligence in terms of a "possible worlds" semantics – i.e. to view them as asking, counterfactually, how an agent *would* perform, hypothetically, on a series of tests (the tests being goals, defined in relation to environments and reward signals).

Real-world intelligent agents don't normally operate in terms of explicit goals and rewards; these are abstractions that we use to think about intelligent agents. However, this is no objection to characterizing various sorts of intelligence in terms of counterfactuals like: how would system S operate if it were trying to achieve this or that goal, in this or that environment, in order to seek reward? We can characterize various sorts of intelligence in terms of how it can be inferred an agent would perform on certain tests, even though the agent's real life does not consist of taking these tests.

This conceptual approach may seem a bit artificial, but, we don't currently see a better alternative, if one wishes to quantitatively gauge intelligence (which is, in a sense, an "artificial" thing to do in the first place). Given a real-world agent X and a mandate to assess its intelligence, the obvious alternative to looking at possible worlds in the manner of the above definitions, is just looking *directly* at the properties of the things X has achieved in the real world during its lifespan. But this isn't an easy solution, because it doesn't disambiguate which aspects of X 's achievements were due to its own actions versus due to the rest of the world that X was interacting with when it made its achievements. To distinguish the amount of achievement that X "caused" via its own actions requires a model of causality, which is a complex can of worms in itself; and, critically, the standard models of causality also involve counterfactuals (asking "what would have been achieved in this situation if the agent X hadn't been there", etc.) (MW07). Regardless of the particulars, it seems impossible to avoid counterfactual realities in assessing intelligence.

The approach we suggest – given a real-world agent X with a history of actions in a particular world, and a mandate to assess its intelligence – is to introduce an additional player, an *inference agent* δ , into the picture. The agent π modeled above is then viewed as π_X : the model of X that δ constructs, in order to explore X 's inferred behaviors in various counterfactual environments. In the test situations embodied in the definitions of pragmatic and efficient pragmatic general intelligence, the environment gives π_X rewards, based on specifically configured goals. In X 's real life, the relation between goals, rewards and actions will generally be significantly subtler and perhaps quite different.

We model the real world similarly to the "fantasy world" of the previous section, but with the omission of goals and rewards. We define a *naturalistic* context as one in which all goals and rewards are constant, i.e. $g_i = g_0$ and $r_i = r_0$ for all i . This is just a mathematical convention for stating that there are no precisely-defined external goals and rewards for the agent. In

a naturalistic context, we then have a situation where agents create actions based on the past history of actions and perceptions, and if there is any relevant notion of reward or goal, it is within the cognitive mechanism of some agent. A *naturalistic agent* X is then an agent π which is restricted to one particular naturalistic context, involving one particular environment μ (formally, we may achieve this within the framework of agents describe above via dictating that X issues constant "null actions" a_0 in all environments except μ).

Next, we posit a metric space (Σ_μ, d) of naturalistic agents defined on a naturalistic context involving environment μ , and a subspace $\Delta \in \Sigma_\mu$ of inference agents, which are naturalistic agents that output predictions of other agents' behaviors (a notion we will not fully formalize here). If agents are represented as program trees, then d may be taken as edit distance on tree space (Bil05). Then, for each agent $\delta \in \Delta$, we may assess

- the prior probability $\theta(\delta)$ according to some assumed distribution θ
- the effectiveness $p(\delta, X)$ of δ at predicting the actions of an agent $X \in \Sigma_\mu$

We may then define

Definition 4. *The inference ability of the agent δ , relative to μ and X , is*

$$q_{\mu, X}(\delta) = \theta(\delta) \frac{\sum_{Y \in \Sigma_\mu} \text{sim}(X, Y) p(\delta, Y)}{\sum_{Y \in \Sigma_\mu} \text{sim}(X, Y)}$$

where sim is a specified decreasing function of $d(X, Y)$, such as $\text{sim}(X, Y) = \frac{1}{1+d(X, Y)}$.

To construct π_X , we may then use the model of X created by the agent $\delta \in \Delta$ with the highest inference ability relative to μ and X (using some specified ordering, in case of a tie). Having constructed π_X , we can then say that

Definition 5. *The inferred pragmatic general intelligence (relative to ν and γ) of a naturalistic agent X defined relative to an environment μ , is defined as the pragmatic general intelligence of the model π_X of X produced by the agent $\delta \in \Delta$ with maximal inference ability relative to μ (and in the case of a tie, the first of these in the ordering defined over Δ). The inferred efficient pragmatic general intelligence of X relative to μ is defined similarly.*

This provides a precise characterization of the pragmatic and efficient pragmatic intelligence of real-world systems, based on their observed behaviors. It's a bit messy; but the real world tends to be like that.

Intellectual Breadth: Quantifying the Generality of an Agent's Intelligence

We turn finally to a related question: How can one quantify the degree of generality that an intelligent agent possesses? There has been much qualitative discussion of "General AI" or "Artificial General Intelligence," versus "Narrow AI" (GP05), and intelligence

as we have formalized it here is specifically a variety of general intelligence, but we have not yet tried to quantify the notion of generality versus narrowness.

Given a triple (μ, g, T) , and a set Σ of agents, one may construct a fuzzy set $Ag_{\mu,g,T}$ gathering those agents that are intelligent relative to the triple ; and given a set of triples, one may also define a fuzzy set Con_{π} gathering those triples with respect to which a given agent π is intelligent. The relevant formulas are:

$$\chi_{Ag_{\mu,g,T}}(\pi) = \chi_{Con_{\pi}}(\mu, g, T) = \sum_Q \frac{\eta_{\mu,g,T}(Q) V_{\mu,g,T}^{\pi}}{Q}$$

One could make similar definitions leaving out the computational cost factor Q , but we suspect that incorporating Q is a more promising direction. We then propose

Definition 6. *The intellectual breadth of an agent π , relative to the distribution ν over environments and the distribution γ over goals, is*

$$H(\chi_{Con_{\pi}}^P(\mu, g, T))$$

where H is the entropy and

$$\chi_{Con_{\pi}}^P(\mu, g, T) =$$

$$\frac{\nu(\mu)\gamma(g, \mu)\chi_{Con_{\pi}}(\mu, g, T)}{\sum_{(\mu', g', T')} \nu(\mu')\gamma(g', \mu')\chi_{Con_{\pi}}(\mu', g', T')}$$

is the probability distribution formed by normalizing the fuzzy set $\chi_{Con_{\pi}}((\mu, g, T))$.

A similar definition of the intellectual breadth of a context (μ, g, T) , relative to the distribution σ over agents, may be posited. A weakness of these definitions is that they don't try to account for dependencies between agents or contexts; perhaps more refined formulations may be developed that account explicitly for these dependencies.

Note that the intellectual breadth of an agent as defined here is largely independent of the (efficient or not) pragmatic general intelligence of that agent. One could have a rather (efficiently or not) pragmatically generally intelligent system with little breadth: this would be a system very good at solving a fair number of hard problems, yet wholly incompetent on a larger number of hard problems. On the other hand, one could also have a terribly (efficiently or not) pragmatically generally stupid system with great intellectual breadth: this would be a system that was roughly equally dumb in all the contexts under study.

Thus, one can characterize an intelligent agent as "narrow" with respect to distribution ν over environments and the distribution γ over goals, based on evaluating it as having low intellectual breadth. A "narrow AI" relative to ν and γ would then be an AI agent with a relatively high efficient pragmatic general intelligence but a relatively low intellectual breadth.

Conclusion

Our goal here has been to push the formal understanding of intelligence in a more pragmatic direction. More work remains to be done, e.g. in specifying the environment, goal and efficiency distributions relevant to real-world systems, but we believe that the ideas presented here constitute nontrivial progress.

If the line of research pursued here succeeds, then eventually, one will be able to do AGI research as follows: Specify an AGI architecture formally, and then use the mathematics of general intelligence to derive interesting results about the environments, goals and hardware platforms relative to which the AGI architecture will display significant pragmatic or efficient pragmatic general intelligence, and intellectual breadth.

References

- Philip Bille. A survey on tree edit distance and related problems. *Theoretical Computer Science*, 337, 2005.
- Ben Goertzel. *The Structure of Intelligence*. Springer, 1993.
- Ben Goertzel. The embodied communication prior. In *Yingxu Wang and George Baciuc (eds), Proceedings of ICCI-09, Hong Kong*, 2009.
- Ben Goertzel and Cassio Pennachin. *Artificial General Intelligence*. Springer, 2005.
- Marcus Hutter. *Universal AI*. Springer, 2005.
- Shane Legg and Marcus Hutter. A collection of definitions of intelligence. In *Ben Goertzel and Pei Wang (eds), Advances in Artificial General Intelligence*. IOS, 2007.
- Shane Legg and Marcus Hutter. A formal measure of machine intelligence. In *Proceedings of Benelam-2006, Ghent*, 2007.
- Stephen Morgan and Christopher Winship. *Counterfactuals and Causal Inference*. Cambridge University Press, 2007.
- Pei Wang. *Rigid Flexibility: The Logic of Intelligence*. Springer, 2006.

A General Intelligence Oriented Architecture for Embodied Natural Language Processing

Ben Goertzel¹ & Cassio Pennachin¹ & Samir Araujo¹ & Fabricio Silva¹
& Murilo Queiroz¹ & Ruiting Lian² & Welter Silva¹ & Michael Ross & Linas Vepstas¹ & Andre Senna¹

¹ Novamente LLC

1405 Bernerd Place, Rockville MD 20851

² Artificial Brain Laboratory

Xiamen University, Xiamen, China

Abstract

A software architecture is described which enables a virtual agent in an online virtual world to carry out simple English language interactions grounded in its perceptions and actions. The use of perceptions to guide anaphor resolution is discussed, along with the use of natural language generation to answer simple questions about the observed world. This architecture has been implemented within the larger PetBrain system, which is built on the OpenCog open-source AI software framework and architected based on the OpenCogPrime design for integrative AGI, and has previously been used for nonlinguistic intelligent behaviors such as imitation and reinforcement learning.

Introduction

One key requirement of a humanlike AGI is the ability to communicate linguistically about the world it experiences, and to use its world-experience to understand the language others produce. We describe here an approach to achieving these abilities, which has been implemented within the PetBrain software system. After reviewing the current version, we discuss extending it into a more powerful AGI system.

The PetBrain implements a portion of OpenCogPrime (Goe09), an integrated conceptual and software design aimed at achieving roughly humanlike AGI at the human level and ultimately beyond. It is implemented within the OpenCog open-source AI software framework (GH08); and its purpose is to control a virtual pet (currently a dog) in the Multiverse or RealX-Tend virtual worlds. Previous papers have described the PetBrain's capability for imitative and reinforcement learning (GPG08) and its personality and emotion model (GPGA08); here we focus on its capability for linguistic interaction, with a focus on the way its virtual embodiment allows it to resolve linguistic ambiguity and answer questions about itself and its life. The paper is best read in conjunction with two online videos illustrating the phenomena described ¹, ².

¹http://novamente.net/example/grab_ball.html

²<http://novamente.net/example/nlp.html>

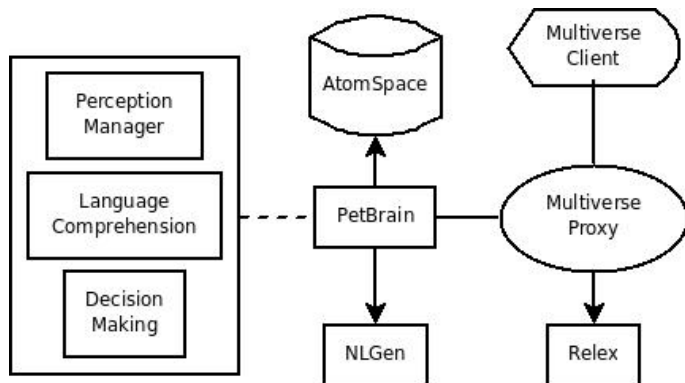


Figure 1: High-level overview of PetBrain software architecture

OpenCog and the PetBrain

OpenCogPrime is a cognitive architecture intended for implementation within the OpenCog AGI software framework, motivated by human cognitive science and overlapping significantly with Stan Franklin's LIDA (FF08) and Joscha Bach's MicroPsi (Bac09) architectures. The architecture consists of a division into a number of interconnected functional units corresponding to different specialized capabilities such as perception, motor control and language, and also an attentional focus unit corresponding to intensive integrative processing. Within each functional unit, knowledge representation is enabled via an AtomSpace software object that contains nodes and links (collectively called Atoms) of various types representing declarative, procedural and episodic knowledge both symbolically and subsymbolically. (For a description of the node and link types typically utilized in OpenCog, the reader is referred to (GP06); here we will mention a few node and link types in passing, assuming the essential semantics will be clear from context.) Each unit also contains a collection of MindAgent objects implementing cognitive, perception or action processes that act on this AtomSpace, and/or interact with the outside world.

The PetBrain, roughly depicted in Figure 1, is a sub-

set of the OpenCogPrime architecture, implemented within OpenCog. Currently it is used to control virtual pets, but in fact it could be used more generally to control various intelligent virtual agents; and work is underway to customize it to control humanoid robots as well (GdG98). The PetBrain stores all its knowledge inside the Atomspace. Part of this knowledge is produced by the agent's (pet's) sensors (exteroceptive and proprioceptive) and handled by the Perception Manager component. The agent's knowledge about the whole environment is used by the Language Comprehension component to link the elements, mentioned in the sentences heard by the agent, to the objects observed by the agent in the virtual world. An agent can recognize and execute commands requested by another agent/avatar, besides answering questions.

Most of our work with the PetBrain to date has involved the Multiverse³ virtual world, though we have also worked with RealXTend. We've customized the Multiverse Server and created a Multiverse Proxy to mediate communication between the Virtual World and the PetBrain. The Multiverse Proxy sends perceptual data from Multiverse to the PetBrain, and in the current system it also sends linguistic relationship. The RelEx language comprehension system is used by the Multiverse Proxy to parse the sentences given by the user, via the Multiverse Client, and only the Relation objects produced by RelEx are then sent to the PetBrain. Conversely, the PetBrain sends Relation objects based on conceptual Atoms to the Multiverse Proxy, which invokes the NLGen system to transform these into English to be conveyed to the user.

The current PetBrain architecture is not intended as a human(or animal)-level AGI but rather as an interesting proto-AGI software system incorporating multiple components designed with human-level AGI in mind. Strategies for incrementally modifying the PetBrain into a human-level AGI will be discussed below.

Natural Language Processing with RelEx and NLGen

OpenCog's current Natural Language Processing subsystem (invoked by the PetBrain but also heavily used outside it) contains two main components: RelEx (GGP⁺06), which is the natural language comprehension engine, takes sentences and maps them into abstract logical relations which can be represented in the OpenCog Atomspace; and NLGen, a natural language generation engine, that translates Atoms embodying logical relations into English sentences.

RelEx itself contains multiple components, only the largest or most pertinent of which will be reviewed here. RelEx carries out syntactic parsing via the open-source Link Parser created at Carnegie-Mellon University (ST91)⁴. It then contains semantic interpretation code that converts the Link Parser output to a feature

structure representation (a directed graph), and uses a series of hand-coded rules ("sentence algorithms") to modify the feature structure. The modified feature structures are used to generate the RelEx semantic relationships corresponding to a sentence, which bear some resemblance to the output of the Stanford dependency parser⁵, but often contain significant additional semantic information. Finally, the RelEx2Frame component uses additional hand-coded rules to map RelEx semantic relationships into sets of more abstract logical relationships, constructed utilizing the FrameNet (BFL98) ontology and other similar semantic resources (Goe08); and the Frame2Atom component translates these relationships into OpenCogAtoms. Thus, RelEx translates English into Atoms. Among the many details we have left out in this precis are the ranking of multiple outputs (parsing and FrameNet interpretations are ranked using a combination of inference and heuristics) and the handling of ambiguity (e.g. anaphor resolution which will be discussed below).

NLGen is a sentence generation system which is used to generate sentences from RelEx semantic relationships – which in turn may be produced by applying a component called Frames2RelEx to appropriate Atoms in the OpenCog Atomspace. One of the core ideas underlying NLGen is that most language generation may be done by reversing previously executed language comprehension processes (GPI⁺10),(LGL⁺10). Given a set of interconnected Atoms to express, NLGen iterates through the predicates P in this set, and for each one it produces a graph G_P of associated RelEx semantic relationships, and then matches this graph against its memory (which stores previously perceived sentences and their semantic interpretations) via the SAGA matching algorithm (TMS⁺07), looking for remembered sentences that gave rise to semantic relationship-sets similar to G_P . The results from doing this matching for different graphs G_P are then merged, and some rules are applied for handling phenomena like tense and determiners, ultimately yielding one or more sentences based on combining (instantiated abstractions of) pieces of the remembered sentences corresponding to selected predicates P . Finally, while this similarity matching approach has quite broad applicability, for dealing with complex sentences it must be augmented by additional mechanisms, and a prototype exists that uses an implementation of Chomsky's Merge operator for this purpose (operating on the same RelEx relationships as the primary NLGen system).

Embodiment-Based Anaphor Resolution

One of the two ways the PetBrain currently relates language processing to embodied experience is via using the latter to resolve anaphoric references in text produced by human-controlled avatars.

In our current work, the PetBrain controlled agent lives in a world with many objects, each one with their

³<http://www.multiverse.net>

⁴<http://www.link.cs.cmu.edu/link/>

⁵<http://nlp.stanford.edu/software/lex-parser.shtml>

own characteristics. For example, we can have multiple balls, with varying colors and sizes. We represent this in the OpenCog Atomspace via using multiple nodes: a single ConceptNode to represent the concept "ball", a WordNode associated with the word "ball", and numerous SemeNodes representing particular balls. There may of course also be ConceptNodes representing ball-related ideas not summarized in any natural language word, e.g. "big fat squishy balls," "balls that can usefully be hit with a bat", etc.

As the agent interacts with the world, it acquires information about the objects it finds, through perceptions. The perceptions associated with a given object are stored as other nodes linked to the node representing the specific object instance. All this information is represented in the Atomspace using FrameNet-style relationships (exemplified in the next section).

When the user says, e.g., "Grab the red ball", the agent needs to figure out which specific ball the user is referring to – i.e. it needs to invoke the Reference Resolution (RR) process. RR uses the information in the sentence to select instances and also a few heuristic rules. Broadly speaking, Reference Resolution maps nouns in the user's sentences to actual objects in the virtual world, based on world-knowledge obtained by the agent through perceptions.

In this example, first the brain selects the ConceptNodes related to the word "ball". Then it examines all individual instances associated with these concepts, using the determiners in the sentence along with other appropriate restrictions (in this example the determiner is the adjective "red"; and since the verb is "grab" it also looks for objects that can be fetched). If it finds more than one "fetchable red ball", an heuristic is used to select one (in this case, it chooses the nearest instance).

The agent also needs to map pronouns in the sentences to actual objects in the virtual world. For example, if the user says "I like the red ball. Grab it," the agent must map the pronoun "it" to a specific red ball. This process is done in two stages: first using anaphor resolution to associate the pronoun "it" with the previously heard noun "ball"; then using reference resolution to associate the noun "ball" with the actual object.

The subtlety of anaphor resolution is that there may be more than one plausible "candidate" noun corresponding to a given pronouns. RelEx's standard anaphor resolution system is based on the classical Hobbs algorithm(Hob78). Basically, when a pronoun (it, he, she, they and so on) is identified in a sentence, the Hobbs algorithm searches through recent sentences to find the nouns that fit this pronoun according to number, gender and other characteristics. The Hobbs algorithm is used to create a ranking of candidate nouns, ordered by time.

We improve the Hobbs algorithm results by using the agent's world-knowledge to help choose the best candidate noun. Suppose the agent heard the sentences:

"The ball is red."

"The stick is brown."

and then it receives a third sentence

"Grab it.".

the anaphor resolver will build a list containing two options for the pronoun "it" of the third sentence: ball and stick. Given that the stick corresponds to the most recently mentioned noun, the agent will grab it instead of (as Hobbs would suggest) the ball.

Similarly, if the agent's history contains

"From here I can see a tree and a ball."

"Grab it."

Hobbs algorithm returns as candidate nouns "tree" and "ball", in this order. But using our integrative Reference Resolution process, the agent will conclude that a tree cannot be grabbed, so "ball" is chosen.

Embodiment-Based Question Answering

Our agent is also capable of answering simple questions about its feelings/emotions (happiness, fear, etc.) and about the environment in which it lives. After a question is asked to the agent, it is parsed by RelEx and classified as either a truth question or a discursive one. After that, RelEx rewrites the given question as a list of Frames (based on FrameNet⁶ with some customizations), which represent its semantic content. The Frames version of the question is then processed by the agent and the answer is also written in Frames. The answer Frames are then sent to a module that converts it back to the RelEx format. Finally the answer, in RelEx format, is processed by the NLGen module, that generates the text of the answer in English. We will discuss this process here in the context of the simple question "What is next to the tree?", which in an appropriate environment receives the answer "The red ball is next to the tree."

Question answering (QA) of course has a long history in AI (Zhe09), and our approach fits squarely into the tradition of "deep semantic QA systems"; however it is innovative in its combination of dependency parsing with FrameNet and most importantly in the manner of its integration of QA with an overall cognitive architecture for agent control.

Preparing/Matching Frames

In order to answer an incoming question, the agent tries to match the Frames list, created by RelEx, against the Frames stored in its own memory. In general these Frames could come from a variety of sources, including inference, concept creation and perception; but in the current PetBrain they primarily come from perception, and simple transformations of perceptions.

However, the agent cannot use the incoming perceptual Frames in their original format because they lack grounding information (information that connects the

⁶<http://framenet.icsi.berkeley.edu>

mentioned elements to the real elements of the environment). So, two steps are then executed before trying to match the Frames: Reference Resolution (described above) and Frames Rewriting. Frames Rewriting is a process that changes the values of the incoming Frames elements into grounded values. Here is an example, using the standard Novamente/OpenCog indent notation described in (GP06) (in which indentation denotes the function-argument relation, as in Python, and *RAB* denotes the relation *R* with arguments *A* and *B*):

Incoming Frame (Generated by RelEx)

```
EvaluationLink
  DefinedFrameElementNode Color:Color
  WordInstanceNode "red@aaa"
EvaluationLink
  DefinedFrameElementNode Color:Entity
  WordInstanceNode "ball@bbb"
ReferenceLink
  WordInstanceNode "red@aaa"
  WordNode "red"
```

After Reference Resolution

```
ReferenceLink
  WordInstanceNode "ball@bbb"
  SemeNode "ball_99"
```

Grounded Frame (After Rewriting)

```
EvaluationLink
  DefinedFrameElementNode Color:Color
  ConceptNode "red"
EvaluationLink
  DefinedFrameElementNode Color:Entity
  SemeNode "ball_99"
```

Frame Rewriting serves to convert the incoming Frames to the same structure used by the Frames stored into the agent's memory. After Rewriting, the new Frames are then matched against the agent's memory and if all Frames were found in it, the answer is known by the agent, otherwise it is unknown.

Currently if a truth question was posed and all Frames were matched successfully, the answer will be "yes"; otherwise "no". Mapping of ambiguous matches into ambiguous responses is left for a later version.

If the question requires a discursive answer the process is slightly different. For known answers the matched Frames are converted into RelEx format by Frames2RelEx and then sent to NLGen, which prepares the final English text to be answered. There are two types of unknown answers. The first one is when at least one Frame cannot be matched against the agent's memory and the answer is "I don't know". And the second type of unknown answer occurs when all Frames were matched successfully they cannot be correctly converted into RelEx format or NLGen cannot identify the incoming relations. In this case the answer is "I know the answer, but I don't know how to say it".

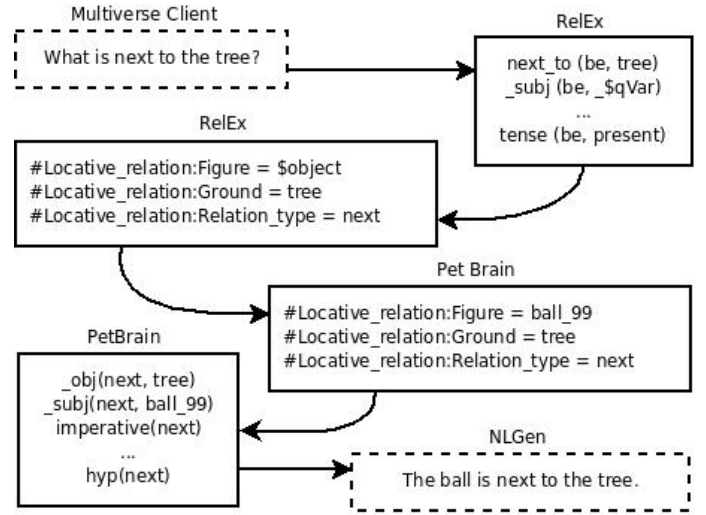


Figure 2: Overview of language comprehension process

Frames2RelEx

As mentioned above, this module is responsible for receiving a list of grounded Frames and returning another list containing the relations, in RelEx format, which represents the grammatical form of the sentence described by the given Frames. That is, the Frames list represents a sentence that the agent wants to say to another agent. NLGen needs an input in RelEx Format in order to generate an English version of the sentence; Frames2RelEx does this conversion.

Currently, Frames2RelEx is implemented as a rule-based system in which the preconditions are the required frames and the output is one or more RelEx relations e.g.

```
#Color(Entity,Color) =>
  present($2) .a($2) adj($2) _predadj($1, $2)
  definite($1) .n($1) noun($1) singular($1)
  .v(be) verb(be) punctuation(.) det(the)
```

where the precondition comes before the symbol => and *Color* is a frame which has two elements: Entity and Color. Each element is interpreted as a variable *Entity* = \$1 and *Color* = \$2. The effect, or output of the rule, is a list of RelEx relations. As in the case of RelEx2Frame, the use of hand-coded rules is considered a stopgap, and for a powerful AGI system based on this framework such rules will need to be learned via experience (a topic beyond the scope of this paper).

Example of the Question Answering Pipeline

Turning to the example "What is next to the tree?", Figure illustrates the processes involved:

The question is parsed by RelEx, which creates the frames indicating that the sentence is a question regarding a location reference (next) relative to an object (tree). The frame that represents questions is called

Questioning and it contains the elements Manner that indicates the kind of question (truth-question, what, where, and so on), Message that indicates the main term of the question and Addressee that indicates the target of the question. To indicate that the question is related to a location, the *Locative_relation* frame is also created with a variable inserted in its element Figure, which represents the expected answer (in this specific case, the object that is next to the tree).

The question-answer module tries to match the question frames in the Atomspace to fit the variable element. Suppose that the object that is next to the tree is the red ball. In this way, the module will match all the frames requested and realize that the answer is the value of the element Figure of the frame *Locative_relation* stored in the Atom Table. Then, it creates location frames indicating the red ball as the answer. These frames will be converted into RelEx format by the RelEx2Frames rule based system as described above, and NLGen will generate the expected sentence "the red ball is next to the tree".

Example of the Language Generation Pipeline

To illustrate the process of language generation using NLGen, as utilized in the context of query response, consider the sentence "The red ball is near the tree". When parsed by RelEx, this sentence is converted to:

```
_obj(near, tree)
_subj(near, ball)
imperative(near)
hyp(near)
definite(tree)
singular(tree)
_to-do(be, near)
_subj(be, ball)
present(be)
definite(ball)
singular(ball)
```

So, if sentences with this format are in the system's experience, these relations are stored by NLGen and will be used to match future relations that must be converted into natural language. NLGen matches at an abstract level, so sentences like "The stick is next to the fountain" will also be matched even if the corpus contain only the sentence "The ball is near the tree".

If the agent wants to say that "The red ball is near the tree", it must invoke NLGen with the above RelEx contents as input. However, the knowledge that the red ball is near the tree is stored as frames, and not as RelEx format. More specifically, in this case the related frame stored is the *Locative_relation* one, containing the following elements and respective values: Figure → red ball, Ground → tree, *Relation_type* → near.

So we must convert these frames and their elements' values into the RelEx format accepted by NLGen. For AGI purposes, a system must learn how to perform this conversion appropriately; currently, however, we have

implemented a temporary short-cut: a system of hand-coded rules, in which the preconditions are the required frames and the output is the RelEx format that will generate the sentence that represents the frames. The output of a rule may contain variables that must be replaced by the frame elements' values. For the example above, the output *_subj(be, ball)* is generated from the rule *output_subj(be, \$var1)* with the *\$var1* replaced by the Figure element value.

Considering specifically question-answering (QA), the PetBrain's Language Comprehension module represents the answer to a question as a list of frames. In this case, we may have the following situations:

- The frames match a precondition and the RelEx output is correctly recognized by NLGen, which generates the expected sentence as the answer;
- The frames match a precondition, but NLGen did not recognize the RelEx output generated. In this case, the answer will be "I know the answer, but I don't know how to say it", which means that the question was answered correctly by the Language Comprehension, but the NLGen could not generate the correct sentence;
- The frames didn't match any precondition; also "I know the answer, but I don't know how to say it" ;
- Finally, if no frames are generated as answer by the Language Comprehension module, the agent's answer will be "I don't know".

If the question is a truth-question, then NLGen is not required: the answer is "Yes" if and only if it was possible to create frames constituting an answer.

From Today's PetBrain to Tomorrow's Embodied AGI

The current PetBrain system displays a variety of interesting behaviors, but is not yet a powerful AGI system. It combines a simple framework for emotions and motivations with the ability to learn via imitation, reinforcement and exploration, and the ability to understand and produce simple English pertaining to its observations and experiences; and shortly it will be able to carry out simple inferences based on its observations and experiences. Assuming the underlying theory is correct, what needs to be done to transform the current PetBrain into an AGI system with, say, the rough general intelligence level of a young child?

Firstly, the current PetBrain QA system has one major and obvious shortcoming: it can only answer questions whose answer is directly given by its experience. To answer questions whose answers are indirectly given by experience, some sort of inference process must be integrated into the system. OpenCog already has a probabilistic engine, PLN (Probabilistic Logic Networks), and one of our next steps will be to integrate it with the PetBrain. For instance, suppose the agent is in a scenario that has many balls in it, of different colors.

Suppose it has previously been shown many objects, and has been told things like "The ball near the tree is Bob's", "the stick next to Jane is Jane's", etc. Suppose that from its previous experience, the agent has enough data to infer that Jane tends to own a lot of red things. Suppose finally that the agent is asked "Which ball is Bob's." The current agent will say "I don't know," unless someone has told it which ball is Bob's before, or it has overheard someone referring to a particular ball as Bob's. But with PLN integrated, then the agent will be able look around, find a red ball and say (for instance) "The ball near the fountain" (if the ball near the fountain is in fact red).

Next, there are known shortcomings in the NLP infrastructure we have used in the PetBrain, some of which have been mentioned above, e.g. the use of hard-coded rules in places where there should be experientially adaptable rules. Remedying these shortcomings is relatively straightforward within the OpenCog architecture, the main step being to move all of these hard-coded rules into the Atomspace, replacing their interpreters with the PLN chainer, and then allowing PLN inference to modify the rules based on experience.

Apart from NLP improvements and PLN integration, what else is missing in the PetBrain, restricting its level of general intelligence? It is missing a number of important cognition components identified in the OpenCog-Prime AGI design. Adaptive attention allocation is prime among these: the ECAN framework (GPI+10) provides a flexible capability for assignment of credit and resource allocation, but needs to be integrated with and adapted to the virtual agent control context. Concept creation is another: inference about objects and linguistic terms is important, but inference becomes more powerful when used in synchrony with methods for creating new concepts to be inferred about. Finally the PetBrain's motivational architecture is overly specialized for the "virtual dog" context and we intend to replace it with a new architecture based on Joscha Bach's MicroPsi (Bac09).

There is also the question of whether virtual worlds like Multiverse are sufficiently rich to enable a young artificial mind to learn to be a powerful AGI. We consider this non-obvious at present, and in parallel with our virtual-worlds work we have been involved with using the PetBrain to control a Nao humanoid robot (GdG98). The OpenCog framework is flexible enough that intricate feedback between robotic sensorimotor modules and cognitive/linguistic modules (such as those described here) can be introduced without changing the operation of the latter.

References

- Joscha Bach. *Principles of Synthetic Intelligence*. Oxford University Press, 2009.
- Collin F. Baker, Charles J. Fillmore, and John B. Lowe. The berkeley framenet project. In *In Proc. of COLING-ACL*, pages 86–90, 1998.
- Stan Franklin and David Friedlander. Lida and a theory of mind. In *Proc. of AGI-08*, 2008.
- Ben Goertzel and Hugo de Garis. Xia-man: An extensible, integrative architecture for intelligent humanoid robotics. In *In Proc. of BICA-08*, pages 86–90, 1998.
- Ben Goertzel, Izabela Freire Goertzel, Hugo Pinto, Mike Ross, Ari Heljakka, and Cassio Pennachin. Using dependency parsing and probabilistic inference to extract relationships between genes, proteins and malignancies implicit among multiple biomedical research abstracts. In *BioNLP '06: Proc. of the Workshop on Linking Natural Language Processing and Biology*, pages 104–111, 2006.
- Ben Goertzel and David Hart. Opencog: An open-source platform for agi. In *Proc. of AGI-08*, 2008.
- Ben Goertzel. A pragmatic path toward endowing virtually-embodied ais with human-level linguistic capability. In *Proc. of IJCNN 2008*, 2008.
- Ben Goertzel. Opencogprime: A cognitive synergy based architecture for embodied general intelligence. In *Yingxu Wang and George Baciuc (eds), Proc. of ICCI-09, Hong Kong*, 2009.
- Ben Goertzel and Cassio Pennachin. The novamente cognition engine. In *Artificial General Intelligence*. Springer, 2006.
- Ben Goertzel, Cassio Pennachin, and Nil Geisweiller. An integrative methodology for teaching embodied non-linguistic agents, applied to virtual animals in second life. In *Proc. of AGI-08*, 2008.
- Ben Goertzel, Cassio Pennachin, Nil Geisweiller, and Samir Araujo. An inferential dynamics approach to personality and emotion driven behavior determination for virtual animals. In *Proc. of Catz and Dogz 2008 (AISB) Symposium*, 2008.
- Ben Goertzel, Joel Pitt, Matthew Ikle, Cassio Pennachin, and Rui Liu. Glocal memory: a design principle for artificial brains and minds. *Neurocomputing, Special Issue of Artificial Brain*, 2010.
- Jerry R. Hobbs. Resolving pronominal references. *Lingua*, 44:311–338, 1978.
- Ruiting Lian, Ben Goertzel, Rui Liu, Michael Ross, Murilo Queiroz, and Linas Vepstas. Sentence generation for artificial brains: a glocal similarity matching approach. *Neurocomputing, Special Issue of Artificial Brain*, 2010.
- Daniel D. K. Sleator and Davy Temperley. Parsing english with a link grammar. In *In Third International Workshop on Parsing Technologies*, 1991.
- Yuanyuan Tian, Richard C. McEachin, Carlos Santos, David J. States, and Jignesh M. Patel. Saga: a subgraph matching tool for biological graphs. *Bioinformatics (Oxford, England)*, 23(2):232–239, Jan. 2007.
- Zhiping Zheng. Bibliography on automated question answering, May 2009. <http://www.answerbus.com/bibliography/index.shtml>.

What we might look for in an AGI benchmark

Brandon Rohrer

Sandia National Laboratories
Albuquerque, NM, USA

Abstract

A benchmark in the field of Artificial General Intelligence (AGI) would allow evaluation and comparison of the many computational intelligence algorithms that have been developed. In this paper I propose that an ideal benchmark would possess seven key characteristics: fitness, breadth, specificity, low cost, simplicity, range, and task focus.

Introduction

As researchers in artificial general intelligence (AGI), we are sometimes asked, “What are you trying to do?” and “How will you know when you’ve done it?” And collectively we are forced to answer that we don’t yet know. (Wan08) This is not for lack of ideas or effort. A reading of Goertzel and Pennachin’s book surveying a broad swath of current AGI research makes it clear that many have thought deeply about the question, (GCP07) but the breadth of our backgrounds and our richness of diversity makes consensus challenging. There have been calls for a technical roadmap (LA09; GAS09) and concrete benchmarks (DOP08). This paper is intended as a contribution to the ongoing benchmark development effort.

Choosing a measurement device for AGI, a benchmark, is the key to answering questions about our aims. A benchmark implies a goal and implicitly contains a success criterion. Benchmarks can focus the efforts of a community; for all its limitations the Turing Test (Tur50) provided a fixed target for an entire subculture of artificial intelligence (AI) researchers, providing them with a common frame of reference and a shared language for efficient communication. An AGI benchmark would allow various approaches to be directly compared, promoting both cooperation and competition, as was seen most recently in large alliances and stiff competition in the race to win the Netflix Prize (Net09). Selecting an appropriate benchmark may greatly accelerate progress in AGI research.

Unfortunately, the selection of a good benchmark is difficult. A closely related problem is found in the assessment of human intelligence. The problem of measuring intelligence in humans is far from solved. While a

number of formal measures exist, such as IQ tests, educational grade point averages, and standardized test scores, their merits are hotly contested. There is no consensus as to whether they are measuring “intelligence,” or even a generally accepted definition of the word itself. There are also informal measures of intelligence, such as publication count or Erdős number in academic communities. It can also be argued that success in some critical endeavor reflects fitness and is an indirect indicator of intelligence. Depending upon one’s peer group, success at a critical endeavor may be represented by one’s salary, number of Twitter followers, or World of Warcraft level. From a biological standpoint, intelligence may be indirectly measured by one’s reproductive fitness: the number of one’s children or sexual partners. Despite (or perhaps due to) the large number of people that have devoted effort to defining a single useful measure of general human intelligence, no consensus has been reached. One complicating factor is that we have a conflict of interest; we may occasionally be guilty of advocating intelligence benchmarks at which we are likely to excel, rather than those which are likely to be the most useful.

Given the historical difficulty in choosing human general intelligence benchmarks, do we have a chance of choosing a non-human intelligence benchmark? We share many of the same challenges. We are no closer to a single definition of the term “intelligence.” There is a profusion of potential measures. And we also may be tempted to advocate benchmarks at which our own systems are likely to excel. If there is one lesson we may learn from the history of human intelligence assessment it is that full consensus may be too ambitious. Our ultimate goals may be better served by choosing several benchmarks that are useful to many of us, rather than waiting until we find a single benchmark that is embraced by all.

This is not to say that any benchmark will do. It will require care not to choose a poor one. For example, performance on non-monotonic reasoning tasks has been proposed as a benchmark for artificial reasoning systems. However, closer examination revealed that human performance on the task was not well characterized, resulting in a machine intelligence benchmark that

was poorly aligned to human intelligence. (EP93) Illogic in human performance is not uncommon. Occasionally in the assessment of risk and reward, humans can be outperformed by rats. (Mlo08) This is not completely surprising. Deductive logic and the expectation maximization are tasks at which computers have outperformed humans for some time. But this example specifically highlights the pitfalls associated with benchmark selection. A benchmark based on reward maximization could result in a scale in which machines progress from human-level intelligence to the intelligence of a rodent.

There have been a number of benchmarks of machine performance that could be considered intelligence measures of a very narrow sort. These include classification datasets for supervised and unsupervised machine learning algorithms, (AN07) some of which contain images. (GHP07) There are also standard simulations on which reinforcement learning (RL) algorithms can compare their performance with each other, such as MountainCar (Moo90) and CartPole (GS93). There are a number of autonomous robotics competitions, which are benchmarks in the sense that they allow quantitative comparisons to be made between robotic systems. These include the robot soccer tournaments RoboCup (The09) and FIRA (FIR09), the autonomous submarine competition of the AUVSI (AUV09), AAAI robot contests, and perhaps best known, DARPA’s driverless navigation Grand Challenges (DAR07). These events have demonstrated that a well-defined challenge can mobilize a large mount of effort and resources (which can be encouraged even further by the addition of several million dollars in prize money).

In the remainder of this paper I will enumerate the characteristics that, in my view, are desirable in an AGI benchmark, and propose a benchmark that meets those requirements. It is my hope that this proposal stimulates further discussion on the topic and contributes to the rapid selection of a provisional machine intelligence measure.

Benchmark criteria

Desirable attributes for an AGI benchmark are summarized in Table 1 and discussed below.

Fitness

A benchmark implies a goal. While it may not always state a goal explicitly, it serves as an optimization criterion, which the research community uses to evaluate and direct its collective efforts. A useful benchmark will accurately reflect the goals of those subscribing to it. This may seem too obvious to merit attention, but it is surprisingly easy to pick a benchmark that does not fit this requirement. One purely hypothetical example of this might be found in a corporate environment where health and safety are high priorities. In order to reflect the importance placed on employee well-being, the number of reported injuries might be a reasonable

Table 1: Characteristics of a useful AGI benchmark

Fitness	Success on the benchmark solves the right problem.
Breadth	Success on the benchmark requires breadth of problem solving ability
Specificity	The benchmark produces a quantitative evaluation.
Low Cost	The benchmark is inexpensive to evaluate.
Simplicity	The benchmark is straightforward to describe.
Range	The benchmark may be applied to both primitive and advanced systems.
Task Focus	The benchmark is based on the performance of a task.

choice of a performance benchmark. However, the simplest way to excel on this benchmark is for no employee to perform any work, thus avoiding the possibility of injury. This benchmark fails because it does not represent all the goals of the community, such as survival of the company and employee job satisfaction. However, this particular company is to be applauded for looking past the most common single corporate benchmark: stock price.

An AGI benchmark should reflect the goals of the AGI community. This will be challenging because those goals have not yet been agreed upon, leaving us without a clear target. However there have been a number of specific ideas proposed. (GAS09) The process of benchmark selection may accelerate and sharpen that discussion.

Another possible benefit of choosing a benchmark is that it may actually free us up from trying to extrapolate the results of our research out to a 10 or 50 year goal. We may be able to choose a benchmark that defines a research direction and let the end result be an emergent property of the researchers in our community each performing a local optimization: maximization against the benchmark. This approach may actually be more appropriate than defining a specific long-term goal at the outset. The research process is inherently uncertain and unpredictable. Having an emergent end goal would require a good deal of confidence in the benchmark, but would allow us to make progress toward a final goal that is currently beyond our capacity to visualize or articulate.

Breadth

Goertzel, Arel and Scheutz (GAS09) argued strongly for breadth (a very large task space) and accessibility (the attribute of requiring no previous task-specific knowledge) in an AGI benchmark. These two criteria capture a common sense among AGI researchers that a “general” intelligence can solve a more general class of problems than its forbears, and that it is, in a sense,

cheating for this to be done through extensive knowledge engineering or specialized heuristics. Weng introduced a related notion of task breadth that he termed muddiness. (Wen05) The ability to perform a broad set of tasks is a necessary characteristic of any system aspiring to human level intelligence.

The matching of human capability was the essence of the Turing Test and most AGI goal descriptions have been in a similar vein. In approaching such an ambitious problem it has been common practice in artificial intelligence research to reduce the breadth of the tasks while keeping the goal of human-level performance. There are strong temptations to reduce breadth: narrowing the task space and introducing task-specific system knowledge can produce far more eye-catching results and garner more attention, particularly from funding sources. However, our experience now shows that human-level performance in a narrow area, such as medical diagnoses or playing chess, does not necessarily generalize to a broader task set. Instead, it appears that maintaining breadth will ultimately be the more productive way to approach our long term goals. Keeping benchmarks broad while incrementally increasing performance expectations mimics the process followed by evolution during the development of animal intelligence. It is possible that following this course will automatically prioritize our efforts, focusing them on the most fundamental problems first.

Specificity

A useful benchmark will provide some quantitative measure of a system's value or performance. The best known benchmark from AI, the Turing Test, provides only a binary valuation, pass or fail. A number of similar tests have been proposed that may come closer to capturing the goals of AGI: the Telerobotic Turing Test (GAS09), the Personal Turing Test (CF05), and the Total Turing Test (Har91). Of course a binary benchmark is of limited use if we wish to evaluate systems that are not near the threshold of success. Turing-type tests could be made finer-grained by calibrating them against typical humans of varying ages, rather than setting a single threshold at the performance level of a typical adult. This notion of *cognitive age* (DOP08) could be further extended by calibrating performance against that of other species, resulting in a *cognitive equivalent organism*. A finer-grained measure, rather than a threshold, allows AGI candidates in various stages of development to be compared and progress to be charted over time. It also takes the pressure off researchers to define and come to consensus on a technological roadmap for developing AGI. (GAS09) Instead researchers can let the benchmark drive development priorities. In each particular approach, whatever aspect of technology would have the greatest impact on that system's benchmarked performance, that is where they can focus their efforts. The community would not need to spend time debating whether visual object recognition or non-monotonic logic needs to be addressed most

urgently.

Even more useful would be a benchmark that mapped performance onto a scalar or vector of continuous or finely discretized values. With an appropriate mapping, common distance metrics such as the L^2 norm could be used to rank, order, and describe disparities between multiple AGI candidates. It would still be possible to set a Turing threshold, but a numerical benchmark result would allow evaluation of AGI efforts that fall short of human performance, as well as of those that exceed it.

Low Cost

An ideal benchmark will not require an inordinate amount of time, money, power, or any other scarce resource to evaluate. In order to be useful as a measurement device, it must be practical to apply. Even if it were excellent in all other respects, an incomputable benchmark would be of no practical value.

By taking advantage of economies of scale, competitions have proven to be an efficient way to evaluate a large number of systems in a single event. The overhead of administering the task, constructing the apparatus, and judging the results is shared among all the teams. A benchmark may also be able to use a competition format to reduce its cost in this way.

Simplicity

While not a requirement, it would be desirable for a benchmark to be simple in the sense that it could be accurately and concisely communicated to someone with only a high school (secondary school) diploma. Although the full motivation and justification for the benchmark may be much more complex, the ability to condense the success metric into a brief tagline can do a great deal to promote understanding in the wider scientific and non-scientific communities. This is particularly relevant to potential customers and funding sources. It is much easier to sell an idea if it can be clearly communicated. Simplicity will also promote accurate representation in popular media coverage of AGI. If we are able to provide brief summaries of our goals in the form of a soundbite, we can keep the stories more accurate. Otherwise we risk the distortion and misrepresentation that can inadvertently accompany technical reporting in the popular media.

Range

The best benchmark would be applicable to systems at all stages of sophistication. It would produce meaningful results for systems that are rudimentary as well as for systems that equal or exceed human performance. As was suggested earlier, a benchmark with a wide range of applicability would provide a full roadmap for development, giving direction both for immediate next steps and pointing toward long-range goals. This would have the added benefit of countering critics who might

claim that the goals of AGI are out of reach. A wide-range benchmark would imply near term, concrete goals by which we could measure and report our successes.

Task Focus

The four previous criteria (specificity, low cost, simplicity, and range) point toward a tool-agnostic task-focused benchmark. A performance measure of this type would not explicitly favor any particular approach (connectionist, symbolic, hybrid, or otherwise) but would reward each system purely on its demonstrated merits.

It is uncommon to have a scientific community united and defined by the problem it is trying to solve. It is much more common to have a community built around the use of a single computational, methodological, or modeling tool. This can be useful; it ensures that everyone understands everyone else's work. In a tool-centric community there is a common language and a shared set of assumptions that results in highly efficient communication. It is also easier to define who "belongs". Anyone whose work looks too unusual or unfamiliar is probably using a novel approach and is therefore an outsider.

Despite these benefits, tool-based definition is a luxury the field of AGI can't afford. The last several decades have demonstrated that focus on isolated toolsets is not necessarily the ideal approach to general AI. Any single tool may have hidden inductive biases that, if unacknowledged, can color the interpretation of its results. (THB07) There are now many significant efforts to combine multiple tools, specifically across connectionist-symbolic lines, one of the most notable of which is the DUAL architecture. (Kok94) Although it will require more effort in both explaining our work to each other and in grasping unfamiliar approaches, adopting a methodologically agnostic view greatly increases the size of the net we are casting for solutions. It is also an inoculation against intellectual inbreeding and unexamined assumptions, the primary symptoms of "looking where the light is."

One of the strongest arguments for a tool-centered approach to AGI is the biological plausibility of certain tools. However, this has proven to be a very elastic criterion. For example, artificial neural networks are based on approximate models of some neural circuits, yet some question the biological plausibility of their function. (AA09) Conversely, algorithms with no obvious biological implementation, such as the A* search, can mimic gross aspects of some human behaviors. Our neuroanatomic knowledge is too sparse at this point to conclusively specify or rule out algorithms underlying cognition. Most often the biological plausibility argument serves as a Rorschach test, helping us to expose our technical biases. And although there is some philosophical disagreement on this point among AGI developers, it could be argued that if a machine successfully achieves human-level performance on a broad intelligence metric, the biological plausibility of the approach

is irrelevant.

Biological fidelity is itself an alternative to a task-based benchmark. This is the goal of model-based approaches to AGI. For now, the qualitative nature of biological fidelity makes it an unsatisfying benchmark candidate. Although serious efforts to quantify it are underway (LGW09), they are not yet mature. Interestingly, the proposed framework for establishing biological fidelity is also task-based, with the objective of matching human performance substituted for performance maximization. But until biological fidelity is concretely defined, establishing it more easily takes the form of a legal argument than a scientific one, with no conclusive way to resolve differences of opinion. However, seeking computational insights through biomimicry has been the genesis of many of our current computing tools and will undoubtedly serve as an ever-richer source of inspiration as our understanding of the brain matures.

A task-based benchmark has the additional benefit of keeping claims and counterclaims about competing approaches accurate. Without a mutually accepted basis for comparison, researchers are put in a difficult position when attempting to draw distinctions between their work and that of others. We are often reduced to speculating about the ultimate capabilities and limitations of both our own and others' approaches, a subjective and non-scientific endeavor that is frustrating and can spark animosity. This is an inherently problematic process, as we naturally underestimate those tools with which we are least familiar and overestimate those which we know best, particularly if we helped create them.

It may be reasonably argued that a benchmark with a strong task focus would provide limited support for the development of theory and mathematical analysis. But this is not necessarily the case. Theory and analysis have consistently provided insights that have enhanced performance. The adoption of a task-based benchmark would not make irrelevant rigorous mathematical work on AGI. It would only provide extra motivation to keep such theories grounded. These efforts make powerful mathematical statements about the potential capabilities of inductive problem solvers and thus are highly relevant to AGI. (Hut05; Sch04; Sch09) However, two conditions must be met for these efforts to directly contribute to improving performance on a task-based benchmark. 1) Every mathematical representation of the world makes modeling assumptions. These assumptions must not neglect or distort essential characteristics of the system being modeled. And 2) results must be reducible to practice. If a universal problem solver is mathematically defined, but could not be built with finite resources or run in finite time, it may be of limited value in pursuing a task-based benchmark. Reduction to practice is also a good method to verify that condition 1) was met.

A counter argument could be made that the development of intelligence should center exclusively on ana-

lytical and mathematical problems rather than physical or low-level tasks. The reasoning might be that higher level analytic and cognitive functions are uniquely human and should therefore be the sole focus of any effort to develop human level AI. But the fact remains that whatever cognitive abilities humans have acquired, they were preceded by the phylogenetically more basic abilities used by all mammals to find food, avoid threats, and reproduce. For this reason, more basic tasks of perception and physical interaction should not be neglected in favor of tasks that are more symbolic in nature.

Conclusion

A set of criteria for evaluating AGI benchmarks is proposed in Table 1. This is not intended to be a final answer to how to select a benchmark. Rather it is presented in the spirit of the “straw man,” an imperfect incarnation that invites criticism, suggestions for improvement, and counterproposals. It is hoped that these criteria will promote discussion throughout the community, inspiring new and improved proposals for benchmarks which in turn will bring us closer to achieving our goals by clarifying them.

Acknowledgments

This work was supported by the Laboratory Directed Research and Development program at Sandia National Laboratories. Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy’s National Nuclear Security Administration under Contract DE-AC04-94AL85000.

References

- T. Achler and E. Amir. Neuroscience and AI share the same elegant mathematical trap. In *Proc 2009 Conf on Artificial General Intelligence*, 2009.
- A. Asuncion and D.J. Newman. UCI machine learning repository, 2007. <http://www.ics.uci.edu/~mllearn/MLRepository.html>.
- AUVSI. Auvsi unmanned systems online, 2009. <http://www.auvsi.org/competitions/water.cfm>, Accessed September 22, 2009.
- R. Carpenter and J. Freeman. Computing machinery and the individual: The Personal Turing Test. Technical report, Jabberwacky, 2005. <http://www.jabberwacky.com/personaltt>, Accessed September 22, 2009.
- DARPA. Darpa urban challenge, 2007. <http://www.darpa.mil/grandchallenge/index.asp>, Accessed September 22, 2009.
- W. Duch, R. J. Oentaryo, and M. Pasquier. *Frontiers in Artificial Intelligence Applications*, volume 171, chapter Cognitive architectures: Where do we go from here?, pages 122–136. IOS Press, 2008.
- R. Elio and F. J. Pelletier. Human benchmarks on AI’s benchmark problems. In *Proc 15th Congress of the Cognitive Science Society*, pages 406–411, Boulder, CO, 1993.
- FIRA. Federation of International Robosoccer Association Homepage, 2009. <http://www.fira.net/>, Accessed September 22, 2009.
- B. Goertzel, I. Arel, and M. Scheutz. Toward a roadmap for human-level artificial general intelligence: Embedding HLAI systems in broad, approachable, physical or virtual contexts. Technical report, Artificial General Intelligence Roadmap Initiative, 2009. <http://www.agi-roadmap.org/images/HLAIR.pdf>. Accessed September 21, 2009.
- B. Goertzel and Eds. C. Pennachin. *Artificial General Intelligence*. Springer, 2007.
- G. Griffin, A. Holub, and P. Perona. Caltech-256 object category dataset. Technical Report 7694, California Institute of Technology, 2007. <http://authors.library.caltech.edu/7694>.
- S. Geva and J. Sitte. A cart-pole experiment for trainable controllers. *IEEE Control Systems Magazine*, 13:40–51, 1993.
- S. Harnad. Other bodies, other minds: A machine incarnation of an old philosophical problem. *Minds and Machines*, 1:43–54, 1991.
- M. Hutter. *Universal Artificial Intelligence: Sequential Decisions Based on Algorithmic Probability*. Springer-Verlag, Berlin Heidelberg, 2005.
- B. N. Kokinov. The DUAL cognitive architecture: A hybrid multi-agent approach. In *Proceedings of the Eleventh European Conference on Artificial Intelligence*. John Wiley and Sons, 1994.
- S. Livingston and I. Arel. AGI roadmap, 2009. <http://agi-roadmap.org/>, Accessed September 22, 2009.
- C. Lebiere, C. Gonzales, and W. Warwick. A comparative approach to understanding general intelligence: Predicting cognitive performance in an open-ended dynamic task. In *Proceedings of the Second Conference on Artificial General Intelligence*. Atlantis Press, 2009.
- L. Mlodinow. *The Drunkard’s Walk: How Randomness Rules Our Lives, 8th Printing Edition*. Pantheon, 2008. See Chapter 1.
- A. Moore. *Efficient Memory-Based Learning for Robot Control*. PhD thesis, University of Cambridge, 1990.
- Netflix. Netflix prize homepage, 2009. <http://www.netflixprize.com/>, Accessed September 23, 2009.
- J. Schmidhuber. Optimal ordered problem solver. *Machine Learning*, 54:211–254, 2004.
- J. Schmidhuber. Ultimate cognition à la Gödel. *Cognitive Computing*, 1:177–193, 2009.
- P. Tino, B. Hammer, and M. Bodén. *Perspectives of Neural-Symbolic Integration*, volume 77, chapter 5. Markovian bias of neural-based architectures with

feedback connections, pages 95–133. Springer-Verlag, Heidelberg, Germany, 2007.

The RoboCup Federation. RoboCup Homepage. <http://www.robocup.org/>, 2009. Accessed September 22, 2009.

A. M. Turing. Computing machinery and intelligence. *Mind*, 59:433–460, 1950.

P. Wang. *Frontiers in Artificial Intelligence Applications*, volume 171, chapter What do you mean by AI?, pages 362–373. IOS Press, 2008.

J. Weng. Muddy tasks and the necessity of autonomous mental development. In *Proc. 2005 AAAI Spring Symposium Series, Developmental Robotics Symposium*, Stanford University, Mar 21-23 2005.

A Cognitive Architecture for Knowledge Exploitation

G.W. Ng, Y.S. Tan, L.N. Teow, K.H. Ng, K.H. Tan, R.Z. Chan

Cognition and Fusion Laboratory, DSO National Laboratories, Singapore
{ngeewah,tyuansin,tloonin,nkhinhua,tkhenghw,cruizhon}@dso.org.sg

Abstract

A cognitive architecture specifies a computational infrastructure that defines the various regions/functions working as a whole to produce human-like intelligence [1]. It also defines the main connectivity and information flow between various regions/functions. These functions and the connectivity between them in turn facilitate and provide implementation specifications for a variety of algorithms. Drawing inspirations from Computational Science, Neuroscience and Psychology, a top-level cognitive architecture which models the information processing in human brain is developed. Three key design principles [2] inspired by the brain – Hierarchical Structure, Distributed Memory and Parallelism – are incorporated into the architecture. A prototype cognitive system is developed and it is able to bring to bear different types of knowledge to solve a problem. It has been applied to object recognition in images. The cognitive system is able to exploit bottom up perceptual information, top down contextual knowledge and visual feedback in a way similar to how human utilizes different knowledge to recognize objects in images.

Introduction

A cognitive architecture specifies a computational infrastructure that defines the various regions/functions working as a whole to produce human-like intelligence. It also defines the main connectivity and information flow between various regions/functions. These functions and the connectivity between them in turn facilitate and provide implementation specifications for a variety of algorithms. There exist a number of excellent cognitive architectures but many have overlooked the importance of biological validity.

Many artificial intelligence (AI) techniques and computational theories have been developed over the last few decades. However, the vast majority of them focus on modeling only specific aspects of human intelligence. Hallmarks of human intelligence, such as robustness and adaptability, are usually “programmed” into systems and not as outcomes. To achieve human-like intelligence, we need to look into the seat of human intelligence – the human brain. We need to understand the different parts of the human brain, how they are connected, what kind of information they process and how they process it. Advances in medical science, especially Neuroscience, over the years have allowed us to answer some of these questions. With the help of more sophisticated measuring

devices such as functional Magnetic Resonance Imaging (fMRI), Neuroscience has provided some insights into this area. Although current understanding of the biological aspects of human brain is still quite limited, we can draw inspirations from what can be observed about it. In other words, we can try to model the behaviors of Man, and to a certain extent, the human brain. It is in this aspect that psychology plays a part.

Drawing inspirations from the fields of Computational Science, Neuroscience and Psychology, a top-level cognitive architecture is developed. Various key parts of the human brain and their functions are identified and included in the design. Some of the desired behaviors are set as design principles. The cognitive architecture also models information processing in the human brain. The human brain is able to process information in parallel and is able to bring to bear different types of knowledge, distributed throughout the brain, to solve a problem.

The top-level cognitive architecture design and the design principles will be presented here, together with a description of a prototype cognitive system developed based on this design. This is followed by a discussion on how the cognitive system has been applied to object recognition in images, using contextual knowledge and visual feedback, in a way similar to how a human recognizes objects in images.

Top-level Cognitive Architecture

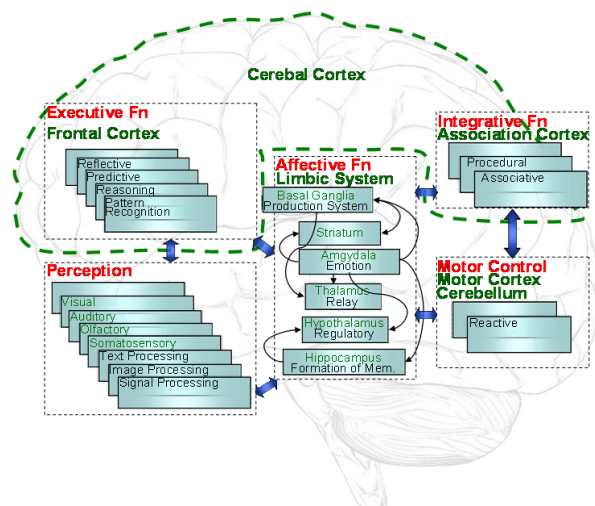


Figure 1: Top-level Cognitive Architecture Design

Core Modules

Five core regions in the human brain, namely, Frontal Cortex, Perception, Limbic System, Association Cortex and Motor Cortex, are identified and shown in Figure 1. Each of these five regions represents a class of functions or processes in the brain. The corresponding classes of functions are Executive Functions, Perception, Affective Functions, Integrative Functions and Motor Control, respectively.

Pre-Frontal Cortex (Executive Functions). The prefrontal cortex (PFC) is the anterior part of the frontal lobes of the brain. It has been implicated in planning complex cognitive behaviors, personality expression, and moderating correct social behavior. It is important when “top-down” processing is needed; that is, when behavior is guided by internal states or intentions [3]. The basic activity of this brain region is considered to be orchestration of thoughts and actions in accordance with internal goals. Executive function relates to abilities to differentiate among conflicting thoughts, determine good and bad, better and best, same and different, future consequences of current activities, working towards a defined goal, prediction of outcomes, expectation based on actions, and social “control”.

Perception. Perception is the process of acquiring, interpreting, selecting, and organizing sensory information.

Limbic System (Affective Functions). The limbic system [4] is a term for a set of brain structures including the hippocampus and amygdala that support a variety of functions including emotion, behavior and formation of long term memory.

Association Cortex (Integrative Functions). John Hughlings Jackson first proposed in the 1870s that the cortex is organized hierarchically and that some cortical areas serve higher-order integrative functions that are neither purely sensory nor purely motor but associative [5]. These higher-order cortices are what we call today the association areas, associating sensory inputs to motor outputs and performing mental task mediating between sensory inputs and motor outputs. Although the association areas are located at various parts of the brain, we have grouped them together as a functional region.

Motor Cortex (Motor Control). It is a term that describes regions of the cerebral cortex involved in the planning, control, and execution of voluntary motor functions.

Key Design Principles

Three main characteristics, Hierarchical Structure, Distributed Memory and Parallelism, of how the human brain works are identified and these characteristics serve as the key design principles for the cognitive architecture. We believe that modeling the different parts of the human

brain and applying these principles will give rise to the robustness, speed, adaptability and other features we have come to associate with human intelligence.

Hierarchical Structure. The neurologist Paul MacLean has proposed that our skull holds not one brain but three [3], each representing a distinct evolutionary stratum that has formed upon the older layer before it, like an archaeological site. He calls it the “triune brain”. He refers to these three brains as the neocortex or neo-mammalian brain, the limbic or paleo-mammalian system, and the reptilian brain that includes the brainstem and cerebellum. Each of the three brains is connected by nerves to the other two, but each seems to operate as its own brain system with distinct capacities.

The archipallium or primitive (reptilian) brain, or “Basal Brain”, called by MacLean the “R-complex” and which includes the brain stem and the cerebellum, is the oldest brain. It consists of the structures of the brain stem - medulla, pons, cerebellum, mesencephalon, and the oldest basal nuclei - the globus pallidus and the olfactory bulbs. In animals such as reptiles, the brain stem and cerebellum dominate. For this reason it is commonly referred to as the “reptilian brain”. It keeps repeating the same behaviors over and over again, never learning from past mistakes. This part of the brain is active, even in deep sleep.

In 1952, MacLean first coined the name “limbic system” for the middle part of the brain. It can also be termed the paleopallium or intermediate (old mammalian) brain. It corresponds to the brain of most mammals, especially the earlier ones. The old mammalian brain residing in the limbic system is concerned with emotions and instincts, feeding, fighting, fleeing, and sexual behavior. To this brain, survival depends on avoidance of pain and repetition of pleasure. Physiologically, it includes the hypothalamus, hippocampus, and amygdala. It has vast interconnections with the neocortex, so that brain functions are neither purely limbic nor purely cortical but a mixture of both. As MacLean understands it, this lowly mammalian brain of the limbic system tends to be the seat of our value judgments, instead of the more advanced neocortex. It decides whether our higher brain has a “good” idea or not, whether it feels true and right.

The Neocortex, alternatively known as the cerebrum, the neopallium, or the superior or rational (neomammalian) brain, comprises almost the whole of the hemispheres (made up of a more recent type of cortex) and some subcortical neuronal groups. It corresponds to the brain of primates and, consequently, the human species. The higher cognitive functions which distinguish Man from the animals are in the cortex. MacLean refers to the cortex as “the mother of invention and father of abstract thought”. In Man, the neocortex takes up two thirds of the total brain mass. Although all other animals also have a neocortex, it is usually relatively small, with few or no folds (indicating the surface area, which is a measure of complexity and development).

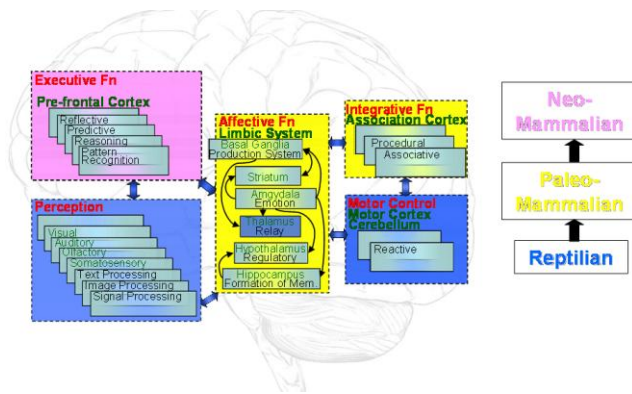


Figure 2: Three levels of Hierarchy in the Human Brain

These three brains form a hierarchy of three brains in one. The cognitive architecture adopts this hierarchical structure in its design to be used as a guide to where various types of knowledge are stored and how information should flow. The various modules in each level of the hierarchy are shown in Figure 2.

Distributed Memory. There are three main types of memory. Semantic Memory consists of facts of the world, disassociated from the place and time when you learned them. Procedural Memory is knowledge about how to do things in the world – it includes your knowledge about how to ride a bicycle, how to type, how to read and understand language, and in general, how to make decisions in selecting actions to achieve goals. Episodic Memory consists of historical episodes or snapshots of specific experiences that are situated in space and time. Studies have shown that memory is not located in any one area in the human brain [6, 7]. Instead, it is distributed throughout the brain. Based on this concept, the cognitive architecture does not have a single module where all the memory or knowledge resides. Each module may have its own memory which it can use to perform its functions or send to other modules when necessary. This will add robustness to the system as it can still function even when some of the functions are down or when knowledge is not complete.

Parallelism. The third key design principle is Parallelism. The human brain does not work in a sequential manner but rather, all the different parts of the brain are constantly running in parallel. This enables the human brain to handle multiple tasks and threads of thoughts at one time. This implies that the brain is able to process different information at the same time. Following this key design principle, the different modules in the cognitive architecture will also be running in parallel. Each module will be developed as an individual running program. The ideal case is to have each of the modules running in one computer in a network. This will allow for true parallelism and hence efficient multi-tasking.

Prototype Cognitive System

A prototype cognitive system (Figure 3) is developed based on the top level design. Some functions from each of the five core regions are developed as modules which form the basic building blocks.

A module is the smallest functional unit of the computational architecture and provides a certain capability. A module is fully encapsulated, with its own knowledge base (distributed long term memory), internal representation schemes and inference methods. Thus a module can be treated like a black box. Other modules in the system do not have to know how it works internally. Each module communicates with other modules either directly or through the Relay (Thalamus) module. Since different modules may have different internal representation schemes, a potential communication problem among the modules may arise in the computational architecture. This problem can be solved by adopting a common representation scheme for all the outputs of the modules.

Modules that perform similar functions are grouped together into classes. For instance, the Perception class comprises of all modules that perform perceptual functions. The reason for grouping similar modules into classes is because different algorithms may be used to find the solution for different problem spaces. By having the concept of classes, each module in the same class can implement just one specific algorithm. This makes the code of each module smaller and easier to maintain. The modules in a class can have complementary, competitive or cooperative relationships. A meta-module for each class may be required to manage the outputs from the different modules within the class.

The prototype system implements each module as an individual executable program. This is in concordance with the parallelism principle of the cognitive architecture.

Description

Perception class: Modules belonging to the Perception class act as receivers to the external world. They take in raw inputs from the external world and process them into useful information. The processed information is then sent to the Relay module for distribution to the rest of the modules in the agent. The current implementation involves a biologically inspired pattern recognition algorithm, Hierarchical Temporal Memory (HTM) [8]. It has an edge over other approaches as it is able to do generalization by exploiting the role of time in vision. In the human eyes, there are short and swift movements called saccades and stops called fixation. We actually make use of these saccades and fixations to visualize and learn the objects we see. This temporal aspect of learning has not been taken into account by many approaches but it is one of the fundamental aspects of HTM that makes it capable of imagery classification.

Motor class: Modules in the Motor class are used to alter both the external environment and the internal state of the

agent. These modules receive instructions from modules such as Selector and apply the necessary actions to the external environment or internal state of the agent.

Association class: Association modules retrieve a list of plausible actions or states when presented with a situation picture. This list of actions or states is associated with the current decision or situation picture. The list is then sent back to the Relay module for further processing by other modules. The current implementation contains a module which builds upon a rule-based engine.

Reasoner class: Reasoner modules analyze situations and proposed actions. They are responsible for higher-level reasoning. The current implementation contains a Dynamic Reasoner module which uses D'Brain [9] for its internal algorithm. D'Brain employs the idea of knowledge fragments and Bayesian reasoning to perform its analysis. The Dynamic Reasoner can be used to fuse different knowledge fragments together.

Selector class: The role of Selector modules is to select an action or a decision from a list of proposed actions or decisions so as to reach the current goals or sub-goals. Currently, the selection process takes into account the probability values provided by the Reasoner modules if they are available. The current implementation contains a FALCON module [10] which enables reinforcement learning in the cognitive system. Reinforcement learning is learning what to do – how to map situations to actions – so as to maximize a numerical reward signal. The learner is not told which actions to take, as in most forms of machine learning, but instead must discover which actions yield the most reward by trying them. Reinforcement learning methods typically have both inductive and deductive aspects: they inductively improve their credibility space on a stage-by stage basis; they deductively select an appropriate response to incoming stimuli using their credibility space. This will enable the Selector module to make better selections over time.

Relay module: The Relay module distributes information to the relevant modules and maintains the current situation picture, in a form of working memory, for all the modules in the system. It functions like the Thalamus in the Limbic System. The current Relay module is able to combine information from different modules and distribute the information to the relevant modules.

Goals Monitoring module: The purpose of the Goals Monitoring module is to produce appropriate sub-goals from the top level goals and then monitor the current situation to check for status of these goals. The status of the goals can be used to update the other modules which may affect their processing of information.

Object Classification in Images

This section will describe how the cognitive system has been applied to object classification in images. Although there has been much research in imagery classification, most algorithms consider each potential target independently and are based solely on measurements of

that target. Due to the nature of the images, the performance of these classification methods generally cannot meet all the operational requirements for accurate classification/recognition.

Human interpreters do not rely solely on the images to do their classification. In reality, they also consider contextual information and inputs from other sources. Hence, regardless of how well the classifier can perform, as long as it does not take into account other information, especially contextual information, users may not have the confidence to use the results of the classification. This is not unlike how we, humans, “classify” objects. We also consider different inputs and contextual information when we are trying to identify objects.

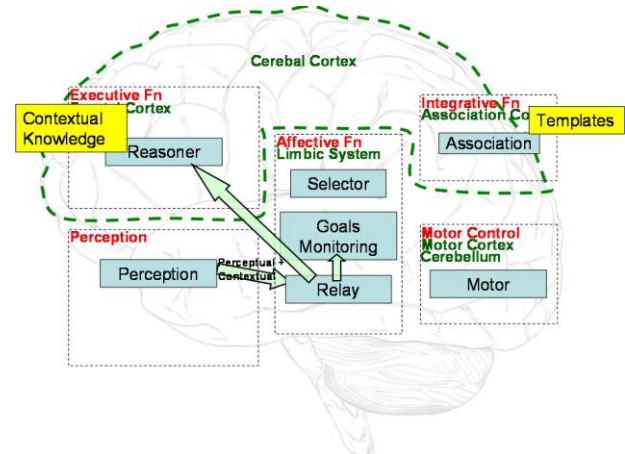


Figure 3: Prototype Cognitive System

Using the Cognitive System

As described previously, the cognitive system is developed based on three key design principles – Parallelism, Hierarchical Structure and Distributed Memory. This leads to certain design features, one of which is the ability to process different kinds of knowledge. This is similar to how Man uses different types of knowledge to solve problems. As mentioned above, there is a need to consider contextual information in the classification process to make it more useful for actual operations. The cognitive system, with its ability to fuse together different types of knowledge, can be used to achieve this.

We, as humans, typically use top-down and bottom-up information to solve problems in a kind of signal-symbol fusion. Contextual knowledge captured in the form of long-term memory is a form of top-down symbolic input while the actual image provides the bottom-up signal information. Contextual knowledge can be seen as a form of prior knowledge which may be learned or gathered through experience. Another top-down information process is feedback to Perception. Previous research has shown that our visual pathways are not unidirectional [11]; in other words, there are also feedback signals to our visual cortex. The system models this by storing templates (the same templates that the Perception module is trained on) in the Association module and retrieving associated templates to send to the Perception module as a visual feedback.

The arrows in Figure 3 show how the perceptual inputs from the image are sent to the different parts of the cognitive system via the Relay module. Certain contextual information may be present in the image itself, for example, a particular formation of objects or other objects in the same image which can help to identify the object of interest. This can be extracted and sent together with the classification results and other contextual information that is outside the image to the other parts of the cognitive system. These form the bottom-up information.

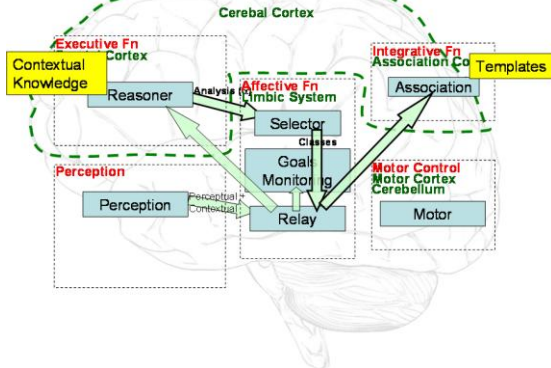


Figure 4: Contextual Information

Contextual knowledge is stored in the Executive Function as shown in Figure 4. The current implementation uses D'Brain as the reasoning engine and the contextual knowledge is captured in the form of Bayesian Network fragments. The HTM output and the contextual information will instantiate some of these fragments which will piece together to form a situation specific Bayesian network. In this way, the bottom-up perceptual inputs are fused with the contextual knowledge. The inference results from the Reasoning engine are then sent to the Selector module. The Selector module will choose the top few classes (classification classes) based on the results and send them to the Association module via the Relay module.

Next, the Association module will retrieve the corresponding templates based on the selected classes. It then sends them to the Perception module, via the Relay module, as feedback to the Perception module. At the Perception module, each template will be “blended” with the original target chip. The blending mechanism is modeled after the human visual recognition process whereby perceived images are adjusted with respect to preconceived templates. Humans model objects and derive the preconceived templates by key features as well as the stabilities of these features. Thus, when we are blending a perceptual input with a potential template, we take into account the features stabilities - features which are more stable are less adjusted. The blended image is then sent to the HTM for classification. This feedback forms part of the top-down information. It is similar to how we retrieve images of objects we have seen before from our long term memory, when we “feel” that the object we are seeing may be one of them. It is important to note here that more than one template is retrieved from the Association module.

When the correct template is used, the feedback should help to boost the confidence that the object belongs to the same class as the template. However, when a template of the wrong class is used, the confidence that the object belongs to this wrong class should be lowered. This can help to prevent the system from being biased to a particular class or self-reinforcing wrongly.

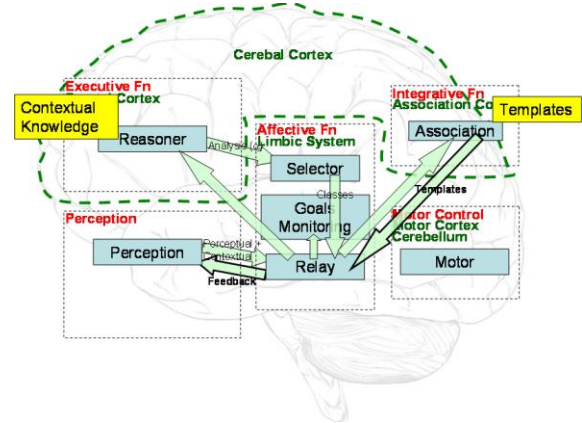


Figure 5: Feedback to the Perception

An Example

An example is used to illustrate how the cognitive system works. The image used for the example is shown in Figure 6. The objective is to identify certain targets in the image. At the start, the user is allowed to enter any contextual information about the image which he may have gathered from other sources. In this example, the user inputs the information that the image is taken near a certain area. The image is passed into the Perception module which carries out a target detection process to find the location of the targets. This is indicated by the square boxes. Next, the Perception module tries to extract whatever contextual information that might exist in the image. The extraction process relies on the contextual knowledge to tell it what to look out for. As formation is one of the contextual knowledge stored, the Perception module tries to see if the targets are deployed in any of known formations. For this case, the targets are found to be deployed in a line formation, as shown by the straight line. Finally, the first target chip on the left is fed into the HTM framework.

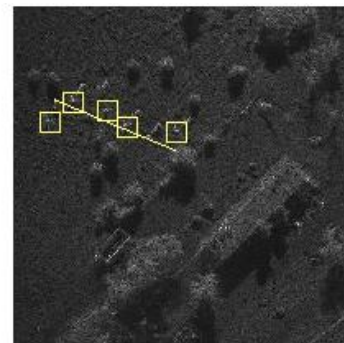


Figure 6: Image

The image processing done to extract the target chip is without the removal of clutter or other image pre-processing. Pre-processing is usually done to “clean up” the image in order to achieve better performance. However, we want to demonstrate how the system can work even when the image quality is far from ideal. As a result, HTM gives a low confidence level of 6% for the correct target class.

Formation	Straight	Random	Circle
Class A	0.150	0.150	0.450
Class B	0.150	0.750	0.400
Class C	0.700	0.100	0.150

Location	Area X	Area Y	Area Z
Class A	0.800	0.100	0.100
Class B	0.100	0.800	0.100
Class C	0.100	0.100	0.800

Figure 7: Contextual Knowledge

This result is sent to the Reasoner module which fuses it with the contextual information by instantiating the corresponding contextual knowledge fragments given in Figure 7. As mentioned, two pieces of contextual information were exploited – there is a formation of five targets deployed in a straight line and secondly, this image was taken near a particular area known to contain a certain type of target. One can treat these two pieces of contextual information as belief inputs on the target class from two experts that are conditioned on the information outside the image. The fusion is based on a series of multiplication operations and renormalization [9]. As a result of considering the contextual information, the reasoning engine is able to increase the confidence level for the correct target class to 46%.

This result is then sent to the Selector module which selects the classes of templates to retrieve from the Association module. The selected templates are then sent to the Perception module where it is blended with the original target chip. The blended image is fed into the same HTM framework. Through blending, the template of the correct class is able to fill up some of the missing gaps in the original image as well remove some of the noise. This helps to push the confidence level for the correct target class up to 67%. Finally, the Reasoner fuses this new output from the Perception module with the contextual information to give a high confidence level of 97%. The system stops here as the top-level goal of achieving at least 80% confidence has been met.

Conclusions

A cognitive architecture that models after the human brain information processing is presented here. It identifies core regions of the human brain and functions that exist in each region. Key design principles inspired by the human brain are discussed and used in the cognitive architecture. It is believed that the hallmarks of human intelligence is an outcome of the way the human brain is designed and the cognitive architecture attempts to reproduce these.

A prototype cognitive system has been developed and described here. Various modules from the cognitive architecture are implemented using existing algorithms and programs. One key feature of the cognitive system is its ability to bring to bear different types of knowledge to solve problems and this is demonstrated with an imagery classification example.

Results show that incorporating contextual information and visual feedback in a human-like approach helps to improve the performance of imagery classification. In the example, the confidence of correct classification increases from 6%, when only the target chip is considered, to 97%, when all information are considered.

Like the human brain, the cognitive system is developed to be a generic intelligent system which has many potential applications. It can be used to perform different tasks by feeding the relevant knowledge to the system. Current work includes applying the cognitive system to Computer Generate Forces and Unmanned Ground Vehicle navigation.

References

- [1] Newell, A. 1990. *Unified Theories of Cognition*. Cambridge, MA: Harvard University Press.
- [2] Ng, G.W. 2009. *Brain-Mind Machinery*. World Scientific.
- [3] Miller E.K., Cohen J.D. 2001. An integrative theory of prefrontal cortex function. *Annu. Rev. Neurosci.* 24: 167-202.
- [4] MacLean, P. D. 1990. *The triune brain in evolution: Role in paleocerebral functions*. New York: Plenum Press.
- [5] Kandel E.R., Schwartz J.H. & Jessell T.M. 2000, *Principles of Neural Science*. Stamford, Conn: McGraw-Hill.
- [6] Lashley, K. S. 1950. In search of the engram. *Society of Experimental Biology*, Symposium 4: 454-482.
- [7] Fuster J.M. 1998. Distributed Memory for Both Long and Short. *Neurobiology of Learning and Memory*. Vol. 70, Issues 1-2: 268-274.
- [8] Hawkins Jeff. 2004. *On Intelligence*. Times Books, New York.
- [9] Ng, G. W., Ng, K. H., Tan, K. H., & Goh, C. H. K. 2006. The Ultimate Challenge of Commander's Decision Aids: The Cognition Based Dynamic Reasoning Machine. In *Proceeding of 25th Army Science Conference*.
- [10] Tan, A.H., Carpenter, G.A., and Grossberg, S. 2007. Intelligence through interaction: Towards a unified theory for learning. In *Proceedings ISNN, LNCS4491*, 1098-1107.
- [11] Edward M. C. 2004. Feedforward, feedback and inhibitory connections in primate visual cortex. *Neural Networks*, Vol. 17, No. 5-6, pp. 625-632.

A (hopefully) Unbiased Universal Environment Class for Measuring Intelligence of Biological and Artificial Systems

José Hernández-Orallo

DSIC, Univ. Politècnica de València,
Camí de Vera s/n, 46020 Valencia, Spain. jorallo@dsic.upv.es

Abstract

The measurement of intelligence is usually associated with the performance over a selection of tasks or environments. The most general approach in this line is called Universal Intelligence, which assigns a probability to each possible environment according to several constructs derived from Kolmogorov complexity. In this context, new testing paradigms are being defined in order to devise intelligence tests which are anytime and universal: valid for both artificial intelligent systems and biological systems, of any intelligence degree and of any speed. In this paper, we address one of the pieces in this puzzle: the definition of a general, unbiased, universal class of environments such that they are appropriate for intelligence tests. By appropriate we mean that the environments are discriminative and that they can be feasibly built, in such a way that the environments can be automatically generated and their complexity can be computed.

Introduction

This paper presents a feasible environment class which can be used to test intelligence of humans, non-human animals and machines. The environment class is developed under the theory presented in (HOD09), which is, in turn, based on (LH07)(HO00)(DH97). This theory presents the first general and feasible intelligence test framework, which should be valid for both artificial intelligent systems and biological systems, of any intelligence degree and speed. The test is not anthropomorphic, is gradual, is anytime and is exclusively based on computational notions, such as Kolmogorov complexity. And it is also meaningful, since it averages the capability of succeeding in different environments. The key idea is to order all the possible action-reward-observation environments by their Kolmogorov complexity and to use this ordering to make a sample. In order to make this feasible (in contrast to (LH07)), in (HOD09) several constraints are imposed on the environments: (1) time is considered, (2) a time-bounded and computable version of Kolmogorov complexity is used, (3) rewards must be balanced, and (4) environments must be sensitive to the agent actions. The environments can then be used to construct adaptive (anytime) tests to evaluate the intelligence of any kind of

agent. The test configures a new paradigm for intelligence measurement which dramatically differs from the current specific-task-oriented and ad-hoc measurement used both in artificial intelligence and psychometrics.

The previous theory, however, does not make the choice for *an* environment class, but just sets some constraints on the kind of environments that can be used. Consequently, one major open problem is to make this choice, i.e., to find a proper (unbiased and feasibly implementable) environment class which follows the constraints. Once this environment class is identified, we can use it to generate environments to run any of the tests variants introduced in (HOD09).

One recurrent problem is that the reference machine for environments is necessarily an arbitrary choice even though Kolmogorov Complexity only differs in a constant when using two different reference machines. But the constant (especially for short tests) is important, since using a specific universal machine could, in the end, constitute a strong bias for some subjects.

Another problem of using an arbitrary universal machine is that this machine can generate environments which are not discriminative. By discriminative we mean that there are different policies which can get very different rewards and, additionally, these good results are obtained by competent agents and not randomly. Note that if we generate environments at random without any constraint, we have that an overwhelming majority of environments will be completely useless to discriminate between capable and incapable agents, since the actions can be disconnected with the reward patterns, with reward being good (or bad) independently of what the agent does.

In (HOD09) a set of properties which are required for making environments discriminative are formally defined, namely that observations and rewards must be sensitive to agent's actions and that environments are balanced, i.e. that a random agent scores 0 in these environments (when rewards range from -1 to 1). This is crucial if we take time into account in the tests because if we leave a finite time to interact with each environment and rewards go between 0 and 1, a very proactive but little intelligent agent could score well (for a thorough discussion on this see (HO09b)). Given these con-

straints, if we decide to generate environments without any constraint and then try to make a post-processing sieve to select which of them comply with all the constraints, we will have a computationally very expensive (or even incomputable) problem. So, the approach taken in this paper is to generate an environment class that ensures that these properties hold. But, we have to be very careful, because we would not like to restrict the reference machine to comply with these properties at the cost of losing their universality (i.e. their ability to emulate or include any computable function).

And finally, we would like the environment class to be user-friendly to the kind of systems we want to be evaluated (humans, non-human animals and machines), but without any bias in favour or against some of them.

According to all this, in this paper we present an operational way to define a universal environment class from which we can effectively generate valid environments, calculate their complexity and consequently derive their probability.

Definition of the Environment Class

The environment class is composed of a cell space and a set of objects that can move inside the space. In this short note, we only enumerate the most relevant traits of the class. For a more formal and complete definition of the class, we refer to (HO09a).

- **Space:** The space is defined as a directed labelled graph of nodes (or vertices), where each node represents a cell and arcs represent actions. The topology of the space can be quite varied. It can include a typical grid, but much more complex topologies too.
- **Objects:** Cells can contain objects. Objects can have any behaviour (deterministic or not), always under the space topology, can be reactive to other agents and can be defined to act with different actions according to their observations. Objects perform one and only one action at each interaction of the environment (except from the special objects Good and Evil, represented by \oplus and \ominus respectively, which can perform several actions in a row). Good and Evil must have the *same* behaviour.
- **Observations and Actions:** Actions allow the evaluated agent (denoted by π) to move in the space. Observations show the (adjacent) cell contents.
- **Rewards:** We will work with the notion of trace and the notion of “cell reward”, that we denote by $r(C_i)$. Initially, $r(C_i) = 0$ for all i . Cell rewards are updated by the movements of \oplus and \ominus . At each interaction, we set 0.5 to the cell reward where \oplus is and -0.5 to the cell reward where \ominus . Each interaction, all the cell rewards are divided by 2. So, an intuitive way of seeing this is that \oplus leaves a positive trace and \ominus leaves a negative trace. The agent π *eats* the rewards it finds in the cells it occupies, updating the accumulated reward $\rho = \rho + r(C_i)$.

The previous environment class is sensitive to rewards and observations (the agent can perform actions in such a way that can affect the rewards and the observations), and it is also balanced (a random agent would have an expected accumulated reward equal to 0). For the formal definition of these properties, see (HOD09). For the proofs of these properties see (HO09a). These properties make the environments suitable for an anytime test (HOD09).

Spaces and objects are coded with Markov algorithms (Turing-complete), their complexity computed and their probability derived. See (HO09a) for details.

Conclusions

Some choices made in this paper can obviously be improved, and better classes might be more elegantly defined. However, to our knowledge, this is the first attempt in the direction of setting a general environment class for intelligence measurement which can be effectively generated and coded.

The main idea for the definition of our environment class has been to separate the space from the objects, and two special symmetric objects are in charge of the rewards, in order to define a class which only includes observation and reward-sensitive environments which are balanced. The space sets some common rules on actions and the objects may include any universal behaviour. This opens the door to social environments.

Acknowledgments

The author thanks the funding from the Spanish Ministerio de Educación y Ciencia (MEC) for projects EXPLORA-INGENIO TIN2009-06078-E, CONSOLIDER-INGENIO 26706 and TIN 2007-68093-C02, and GVA project PROMETEO/2008/051.

References

- D.L. Dowe and A.R. Hajek. A computational extension to the turing test. In *in Proc. of the 4th Conf. of the Australasian Cognitive Science Society*, 1997.
- J. Hernández-Orallo. Beyond the turing test. *J. of Logic, Language and Information*, 9(4):447–466, 2000.
- J. Hernández-Orallo. A (hopefully) unbiased universal environment class for measuring intelligence of biological and artificial systems. Extd. Version. *available at <http://users.dsic.upv.es/proy/anynt/>*, 2009.
- J. Hernández-Orallo. On evaluating agent performance in a fixed period of time. Extd. Version. *available at <http://users.dsic.upv.es/proy/anynt/>*, 2009.
- J. Hernández-Orallo and D.L. Dowe. Measuring universal intelligence: Towards an anytime intelligence test. *Under Review*, <http://users.dsic.upv.es/proy/anynt/>, 2009.
- S. Legg and M. Hutter. Universal intelligence: A definition of machine intelligence. *Minds and Machines*, 17:391–444, 2007.

Towards Practical Universal Search

Tom Schaul and Jürgen Schmidhuber

IDSIA, University of Lugano
Galleria 2, 6900 Manno
Switzerland

Abstract

Universal Search is an asymptotically optimal way of searching the space of programs computing solution candidates for quickly verifiable problems. Despite the algorithm's simplicity and remarkable theoretical properties, a potentially huge constant slowdown factor has kept it from being used much in practice. Here we greatly bias the search with domain-knowledge, essentially by assigning short codes to programs consisting of few but powerful domain-specific instructions. This greatly reduces the slowdown factor and makes the method practically useful. We also show that this approach, when encoding random seeds, can significantly reduce the expected search time of stochastic domain-specific algorithms. We further present a concrete study where Practical Universal Search (PUnS) is successfully used to combine algorithms for solving satisfiability problems.

Introduction

Universal Search is the asymptotically fastest way of finding a program that calculates a solution to a given problem, provided nothing is known about the problem except that there is a fast way of verifying solutions (Lev73). The algorithm has the property that the total time taken to find a solution is $O(t^*)$, where t^* is the time used by fastest program p^* to compute the solution. The search time of the whole process is at most a constant factor larger than t^* ; typically this depends on the encoding length of p^* . The algorithm itself is very simple: It consists in running all possible programs in parallel, such that the fraction of time allocated to program p is $2^{-l(p)}$, where $l(p)$ is the size of the program (its number of bits).

More formally, assume a Turing-complete language L of binary strings that can encode all possible programs in a prefix-free code. Let p^* be the fastest program that solves a problem of problem complexity n . Then $t^* = f(n)$ is the number of time steps p^* needs to compute the solution. Let $l(p^*)$ be the size of p^* in L . Then the algorithmic complexity of Universal Search is $O(f(n))$. However, the multiplicative constant hidden by this notation turns out to be $2^{l(p^*)}$. (All the above assumes that there is a known way of verifying a given

solution to the problem in time linear in the problem size n .)

Searching an infinite number of programs in parallel is impossible on a physical computer, thus an actual implementation of this algorithm has to proceed in phases, where in each phase more and more programs are run in parallel and the total search time per phase is continually increased. See algorithm 1 for the pseudocode.

Algorithm 1: Universal Search.

Input: Programming language, solution verifier
Output: Solution
 $phase := 1$;
while *true* **do**
 for all programs p with $l(p) \leq phase$ **do**
 $timelimit := 2^{phase-l(p)}$;
 run p for maximally $timelimit$ steps;
 if *problem solved* **then**
 return solution;
 end
 end
 $phase := phase + 1$;
end

For certain concrete problems and general-purpose languages it may seem improbable that the fastest program solving the problem can be encoded by fewer than, say, 50 bits, corresponding to a slowdown factor of $2^{50} \approx 10^{15}$, making Universal Search *impractical*.

Previous Extensions and Related Work

Several extensions of universal search have made it more useful in practice. The *Optimal Ordered Problem Solver* (OOPS, (Sch04)) incrementally searches a space of programs that may reuse programs solving previously encountered problems. OOPS was able to learn universal solvers for the Tower of Hanoi puzzle in a relatively short time, a problem other learning algorithms have repeatedly failed to solve. In (Sch95) a probabilistic variant of Universal Search called *Probabilistic Search* uses a language with a small but general instruction set to generate neural networks with exceptional generalization properties.

A non-universal variant (WS96) is restricted to strictly domain-specific instructions plus a jump statement. It is applied successfully to solving partially observable maze problems. The same paper also presents *ALS*, an adaptive version of Universal Search, which adjusts instruction probabilities based on experience.

Another recent development is Hutter’s *HSearch* algorithm (Hut02). *HSearch* combines Universal Search in program space with simultaneous search for proofs about time bounds on their runtime. The algorithm is also asymptotically optimal, but replaces the multiplicative slowdown by an additive one. It may be significantly faster than Universal Search for problems where the time taken to verify solutions is nontrivial. The additive constant depends on the problem class, however, and may still be huge. A way to dramatically reduce such constants in some cases is a universal problem solver called the Gödel Machine (Sch09).

Other attempts have been made at developing practically useful non-exhaustive search algorithms inspired by Universal Search. This family of algorithms include time-allocation algorithms for portfolios of diverse algorithms (GS06).

Making Universal Search Practical

The more domain knowledge we have, the more we can shape or restrict the space of programs we need to search. Here we make Universal Search practically useful by devising a domain-specific language that encodes plausible (according to prior knowledge) programs by relatively few bits, thus reducing the slowdown factor to an acceptable size.

Dropping assumptions

Universal Search makes a number of assumptions about the language L . We will keep the assumption that L is a prefix-free binary code, and drop the following ones:

- L is Turing-complete,
- Every encoding corresponds to a valid program,
- L is infinite.

This does not mean that the opposites of those assumptions are true, only that they are not necessarily true (L is still allowed to be infinite or Turing-complete).

Another implicit assumption that is sometimes made on L is that its encodings represent a sequence of instructions in a standard programming language. Subsequently, we generalize this interpretation to include more restricted languages, such as encodings of parameter settings, random number generator seeds or top-level routines (e.g. `localSearch()`).

Thus, for *Practical Universal Search* (PUnS), L can encode an arbitrary set of programs, all of which can be domain-specific. While the language L thus may become more flexible, the search algorithm for it remains identical to Algorithm 1.

Optimality

PUnS inherits its optimality property directly from Universal Search. As long as the language remains Turing-complete, it has the same asymptotically optimal runtime complexity. In general it will be more restrictive, so this statement does not necessarily hold anymore. Still, the following, weaker one, holds:

Property 1 *For every problem instance, the order of runtime complexity of PUnS is the same as that of the best program which its language can encode.*

Integrating Domain Knowledge

There are two concrete approaches for integrating domain knowledge:

- We can restrict the language, to allow only programs that are appropriate for the problem domain. This can be done in a straightforward way if L is small and finite.
- We can bias the allocation of time towards programs that we suspect to perform better on the problem domain. Universal Search allocates time according to the descriptive complexity (i.e. the number of bits in its encoding) of the program. This is related to the concept of Occam’s Razor, reflecting the hope that shorter programs will generalize better. Now, given domain knowledge about which programs will generally perform better, we can employ the same reasoning and encode those with fewer bits.

Fundamental Trade-off

Defining the language is the key element in PUnS – but this step has a strong inherent (and unresolvable) trade-off: the more general the language, the bigger the slowdown factor, and the more we reduce that one, the more biased the language has to be.

PUnS should therefore be seen as a broad *spectrum* of algorithms, which on one extreme may remain completely universal (like the original Universal Search) and cover all quickly verifiable problems. On the other extreme, if the problem domain is a single problem instance, it may degenerate into a zero-bit language that always runs the same fixed program (e.g. a hand-coded program that we know will efficiently solve the problem). In practice, neither of those extremes is what we want – we want an approach for solving a large number of problems within (more or less) restricted domains. This paper describes a general way of continually adjusting the universality/specificity of PUnS.

Practical Considerations

PUnS is a good candidate for multi-processor approaches, because it is easily *parallelizable*: the programs it runs are independent of each other, so the communication costs remain very low, and the overhead of PUnS is negligible.

Beyond the design of the language L , PUnS has no other internal parameters that would require tuning.

Furthermore, it is highly *tolerant* w.r.t. poorly designed languages and incorrect domain-knowledge: the result can never be catastrophic, as for every problem instance PUnS will still have the runtime complexity of the best solver that the language can express. Thus, an inappropriately designed language can be at most a constant factor worse than the optimal one, given the same expressiveness.

Languages for PUnS

The only condition we need to observe for L is that the encodings remain a prefix-free language. For complete generality, the language can always contain the original Turing-complete language of Universal Search as a fallback. Those encodings are then shifted to higher length, in order to free some of the encodings for the domain-specific programs.

The following sections will describe some variations of PUnS, discussing some specific points along the spectrum (as mentioned above) in more depth. Clearly, if appropriate in a domain, all those types of languages can be combined into a single *hybrid* language.

Domain-biased Programming Languages

Consider a domain where no efficient or general algorithms for solving problems exist, so that it is necessary to search very broadly, i.e. search the space of programs that might solve the problem. If we use a standard Turing-complete language that encodes sequences of instructions, we have more or less the original Universal Search - and thus a huge constant slowdown. However, we can integrate domain knowledge by adding (potentially high-level) domain-specific subroutines with short encodings to bias the search. Furthermore, we can make the language *sparser* by restricting how instructions can be combined (reminiscent of strong typing in standard programming languages). A language like this will remain Turing-complete, and the slowdown factor still risks to be high: the runtime will be acceptable only if the modified language is either very sparse, i.e. almost all bit-strings do not correspond to legal programs and thus only relatively few programs of each length are run¹, or it is compact enough to allow for solution-computing programs with no more than 50 bits. Successfully applied examples of this kind of PUnS can be found in (WS96; Sch95; Sch05).

A language that directly encodes solutions (with a domain-specific complexity measure) causes PUnS to perform a type of exhaustive search that iteratively checks more and more complex solutions. This was explored in a companion paper (KGS10) for searching the space of neural networks, ordered by their encoding length after compression.

¹Note that the cost of finding legal programs dominates when the language is extremely sparse, that is, only solution-computing programs are legal.

Exploration of Parameter Space

If we know a good algorithm for arriving at a solution to a problem, but not the settings that allow the algorithm to solve the problem efficiently (or at all), PUnS can be used to search for good parameters for the algorithm. In this case, each program tested by PUnS is actually the same algorithm, run with different parameters. We can view the interpretation of that language as a non-Turing complete virtual machine that runs “programs” specified as parameter settings.

Any parametrized algorithm could be used as a virtual machine for this type of search. However, the algorithms that are best suited for this purpose are those where parameters are discrete, and can naturally be ordered according to the complexity of the search resulting from a particular parameter setting. There is a wide range of machine learning algorithms that exhibit this characteristic in various ways (e.g. the number of free variables in a function approximator used by an algorithm).

Stochastic Algorithms

Consider a domain where a good algorithm exists and the algorithm is either non-parametric, or good settings for its parameters are known. However, the algorithm is stochastic, and converges to a solution only in a small (unknown) fraction of the runs. In such a domain, universal search could be employed to search the space of random number generator seeds for the algorithm. These seeds are naturally ordered by length, encoded as prefix-free binary integers. While this is a very degenerate language, it fulfills all criteria for being used by Universal Search.

In this case PUnS will spawn more and more processes of the stochastic algorithm in every phase, each with a different seed, until one of them eventually finds a solution. As the encodings have incrementally longer encodings, we do not need to know anything about the probability of success: exponentially more time is allocated to processes with short encodings, so PUnS will only spawn many more processes if they are needed, i.e. if the first random seeds do not lead to convergence fast enough.

In the rest of this section, we will present one such example language, and analyze under which circumstances it is advantageous to apply PUnS to it. Consider the language that encodes an unlimited number of random seeds as 0^k1 for the k th seed, such that seed k is allocated 2^{-k} of the total time.

Let us assume a stochastic base-algorithm where the time T required to find the solution is a random variable, with a probability density function $\phi(t)$ and cumulative probability function $\Phi(t) = P(t_{req} \leq t)$.

Then the time required by PUnS to find the solution T' is the minimum of an infinite number of independent realizations of T , with exponentially increasing penalties:

$$T' = \min(2^1T, 2^2T, 2^3T, \dots, 2^kT, \dots)$$

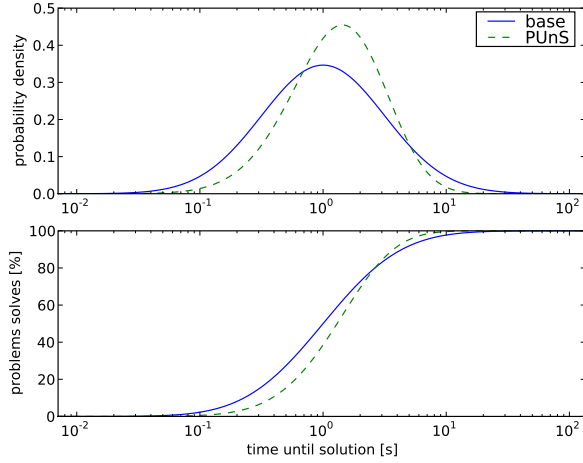


Figure 1: Above: Probability density functions ϕ and ϕ' , of the base distribution ($\sigma = \frac{1}{2} \log(10)$) and PUnS, respectively. Below: percentage of problems solved faster than a certain time, for the base-algorithm and PUnS.

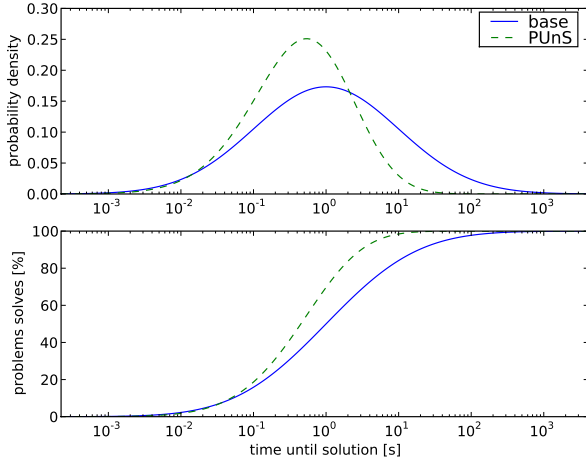


Figure 2: Above: Probability density functions ϕ and ϕ' , of the wider base distribution ($\sigma = \log(10)$) and corresponding PUnS, respectively. Below: percentage of problems solved faster than a certain time, for the base-algorithm and PUnS.

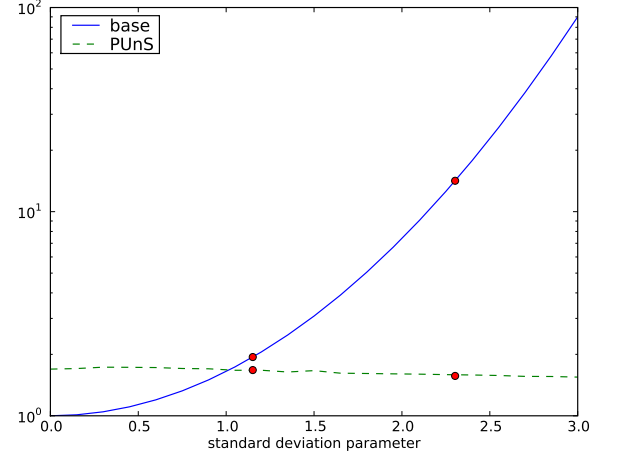


Figure 3: Mean times as a function of the σ parameter, for both the base-algorithm and PUnS. The circles correspond to the values for figures 1 and 2. Note the log-scale on the y-axis: the mean time for the base algorithm increases faster than exponential w.r.t. σ , while it decreases slightly for PUnS.

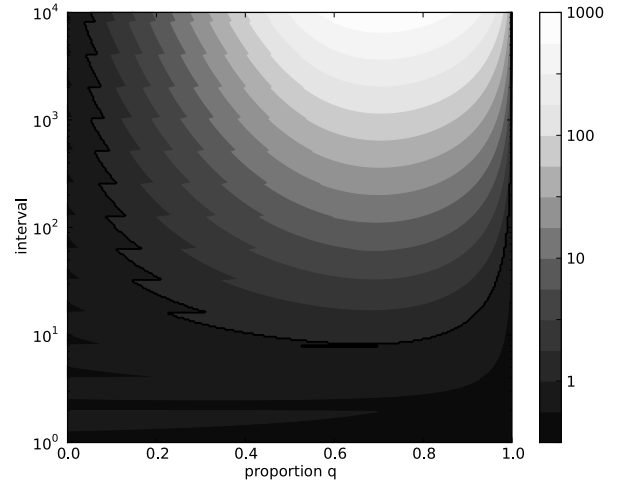


Figure 4: The shades of grey in this plot code for the proportion t_b/t_p , i.e. the factor by which the expected solution time is reduced when employing PUnS instead of the base-algorithm. The horizontal axis shows the dependency on proportion q , while the vertical axis corresponds to the interval size λ . The black line corresponds to limit cases, where both versions have the same expected time: in the whole upper middle part PUnS is better (large enough interval, and not too small q), sometimes by orders of magnitude. The discontinuities (dents) happen whenever λ traverses a power of 2, i.e. whenever k is incremented.

T' has density function

$$\phi'(t) = \sum_{k=1}^{\infty} \phi(t/2^k) \prod_{i=1, i \neq k}^{\infty} 1 - \Phi(t/2^i)$$

and cumulative density function

$$\Phi'(t) = 1 - \prod_{k=1}^{\infty} (1 - \Phi(t/2^k)).$$

Note that it is possible to truncate the computation of the infinite sums and products after a small number of terms, under the reasonable assumption that $\phi(t)$ decays fast as t approaches zero.

Figure 1 illustrates the simple case where the required time is normally distributed in log-time space (i.e. the log-normal distribution) with $\mu = 0$ and $\sigma = \frac{1}{2} \log(10)$. We observe that PUnS reduces the probability of long runtimes (over 5 seconds). In general it has the property of reducing the right (expensive) tail of the base distribution. When the base distribution has a larger standard deviation the effect is even more pronounced (see Figure 2, which shows the same plot as before, but for $\sigma = \log(10)$). In this case we observe an additional beneficial effect, namely that the mean time is reduced significantly. In figure 3 we plot the mean times as a function of σ to illustrate this effect in detail.

Another case of interest is a stochastic base-algorithm with two distinct outcomes: with probability q it finds the solution after t_1 , otherwise it requires $t_2 = \lambda t_1$. This algorithm has an expected solution time of

$$\bar{t}_b = t_1 [1 + (\lambda - 1)(1 - q)].$$

Applying PUnS to the above language, it can be shown that the expected time changes too

$$\bar{t}_p = t_1 \left[1 + (\lambda - 2^k)(1 - q)^{k+1} + \sum_{i=0}^k 2^i (1 - q)^i \right],$$

where $k = \lfloor \log_2 \lambda \rfloor$ is the largest integer such that $2^k \leq \lambda$. Figure 4 shows for which values of q and λ PUnS outperforms the base-algorithm, and by how much (note that those results are independent of t_1).

To summarize, whenever we have access to a stochastic domain-specific algorithm with high variability in its solution times, using PUnS with a simple language to encode random seeds (e.g. the one introduced in this section) can reduce the expected solution time by orders of magnitude.

Case study: SAT-UNSAT

This section presents a small case-study of using PUnS on a mixed SAT-UNSAT benchmark with 250 boolean variables. We use as the underlying base-programs two standard algorithms:

- A local search algorithm (G2WSAT, (LH05)) which is fast on satisfiable instances, but does not halt on unsatisfiable ones.

- A complete solver (Satz-Rand (GSCK00)) that can handle both kinds of instances, but is significantly slower.

Both these algorithms are stochastic, but G2WSAT has a high variance on the time needed to find a solution for a given instance. We set all parameters to default values (G2WSAT: noise = 0.5, diversification = 0.05, time limit = 1h; Satz-Rand: noise = 0.4, first-branching = most-constrained) (GS06).

The language we define for PUnS combines both base-algorithms, employing the coding scheme introduced in the previous section for the high-variance G2WSAT: '11' encodes running of Satz-Rand, '01', '001', '0...01' encode running of G2WSAT with a random seed (a different seed for every number of '0' bits).

In this case, a third of the total computation time is allocated to each Satz-Rand, the first random seed for G2WSAT and all other random seeds combined. With this language, the optimal performance corresponds to that of *Oracle* which for every problem instance knows in advance the fastest solver and the best random seed.

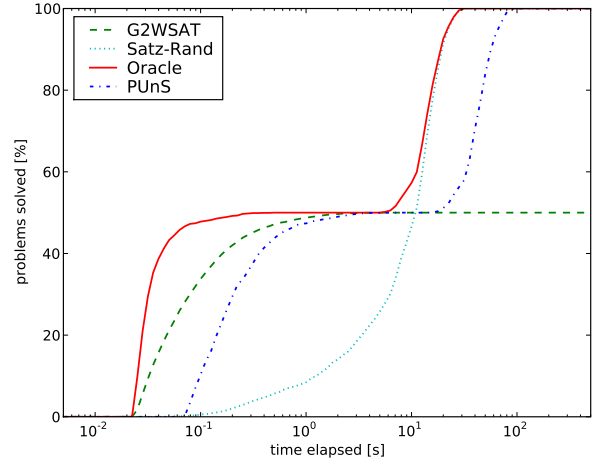


Figure 5: Percentage of instances solved, given a certain computation time for G2WSAT, Satz-Rand, Oracle and PUnS on the mixed SAT-UNSAT-250 benchmark (averaged over 20 runs with different random seeds).

Figure 5 shows the results of running all four algorithms (including Oracle) on a set of 100 satisfiability instances, half of which are unsatisfiable. We find that Practical Universal Search is indeed a robust way of combining the base-algorithms. By construction, it is never slower by more than a factor 3 w.r.t. the best base-algorithm. In addition, the reduced risk of a bad initialization (seed) for G2WSAT on the boundary cases (almost unsatisfiable) is clearly visible as well: Compare the much steeper increase of the PUnS plot, as compared to the G2WSAT one. Finally, as expected, the PUnS performance is approximately that of *Oracle* with a constant factor slowdown – the difference is

due to the fact that the encoding length of the optimal random seed is not bounded a priori.

Conclusions

Universal Search can be used in practice by biasing its language for encoding programs. We provided guidelines for integrating domain-knowledge, possibly (but not necessarily) at the cost of universality. We described a simplified language for non-universal problem domains, and emphasized the flexibility of the approach. In particular, we established that encoding random seeds for stochastic base-algorithms can be highly advantageous. Finally we conducted a proof-of-concept study in the domain of satisfiability problems.

Future work

One direction to pursue would be to develop a general adaptive version of PUnS, where program probabilities change over time based on experience, like in ALS (WS96). A related direction will be to extend PUnS along the lines of OOPS (Sch04), reducing sizes and thus increasing probabilities of encodings of programs whose subprograms have a history of quickly solving previous problems, thus increasing their chances of being used in the context of future problems. There also might be clever ways of adapting the language based on intermediate results of (unsuccessful) runs, in a domain-specific way.

Acknowledgments

We thank Matteo Gagliolo for permission to use his experimental data, as well as Julian Togelius for his valuable input. This work was funded in part by SNF grant number 200021-113364/1.

References

- Matteo Gagliolo and Jürgen Schmidhuber. Learning Dynamic Algorithm Portfolios. *Annals of Mathematics and Artificial Intelligence*, 47(3-4):295–328, August 2006.
- Carla P Gomes, Bart Selman, Nuno Crato, and Henry A Kautz. Heavy-Tailed Phenomena in Satisfiability and Constraint Satisfaction Problems. *Journal of Automated Reasoning*, 24:67–100, 2000.
- Marcus Hutter. The Fastest and Shortest Algorithm for All Well-Defined Problems. *International Journal of Foundations of Computer Science*, 13:431–443, 2002.
- Jan Koutník, Faustino Gomez, and Jürgen Schmidhuber. Searching for Minimal Neural Networks in Fourier Space. In *Proceedings of the Conference on Artificial General Intelligence*, Lugano, Switzerland, 2010.
- Leonid A Levin. Universal sequential search problems. *Problems of Information Transmission*, 9:265–266, 1973.
- Chu M Li and Wenqi Huang. Diversification and Determinism in Local search for Satisfiability. In *Proceedings of the 8th International Conference on Satisfiability (SAT)*, pages 158–172, 2005.
- Jürgen Schmidhuber. Discovering solutions with low Kolmogorov complexity and high generalization capability. In A Prieditis and S Russell, editors, *Proceedings of the International Conference on Machine Learning (ICML)*, pages 488–496, 1995.
- Jürgen Schmidhuber. Optimal Ordered Problem Solver. *Machine Learning*, 54:211–254, 2004.
- Tom Schaul. Evolving a compact, concept-based Sokoban solver, 2005.
- Jürgen Schmidhuber. Ultimate Cognition à la Gödel. *Cognitive Computation*, 1:177–193, 2009.
- Marco Wiering and Jürgen Schmidhuber. Solving POMDPs using Levin search and EIRA. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 534–542, 1996.

Software Design of an AGI System Based on Perception Loop

Antonio Chella

Dip. di Ingegneria Informatica
University of Palermo
chella@dinfo.unipa.it

Massimo Cossentino

ICAR-CNR
Consiglio Nazionale delle Ricerche
cossentino@pa.icar.cnr.it

Valeria Seidita

Dip. di Ingegneria Informatica
University of Palermo
seidita@dinfo.unipa.it

Abstract

According to the *externalist* approach, subjective experience hypothesizes a processual unity between the activity in the brain and the perceived event in the external world. A perception loop therefore occurs among the brain's activities and the external world. In our work the metaphor of *test* is employed to create a software design methodology for implementing an AGI system endowed with the perception loop. Preliminary experiments with a NAO humanoid robots are reported.

Position Statement

Machine consciousness concerns the attempt to model and implement in a robot those aspects of human cognition which are identified with the controversial phenomenon of consciousness, see (CM09) for a review. It does not necessarily try to reproduce human consciousness as such, insofar as human consciousness could be unique due to a complex series of cultural, social, and biological conditions. However, many authors suggested one or more specific aspects and functions of consciousness that could, at least in principle, be replicated in a robot.

It should be remembered that at the beginning of the information era there was no clear cut separation between intelligence and consciousness. Both were considered vaguely overlapping terms referring to what the mind was capable of. There was no clear boundary between intelligence and consciousness.

Today, machine consciousness assumes that there is some aspect of the mind that has not yet been adequately addressed. Therefore, scholars in the field suspect that there could be something more going on in the mind than what is currently under scrutiny in field such as artificial intelligence, cognitive science, and computer science. AGI researchers would agree that there is still a lot of work to do: better algorithms, more data, more complex, and faster learning structures. However it could be doubted whether these improvements in AGI would ever lead to an artificial agent equivalent to a biological mind or it would rather miss some necessary aspect. We, among others, suspect that classic AI missed something important; put it more clearly, we

claim that facing consciousness, and in particular the subjective experience, is an essential requirement for an AGI based artifact, see, e.g. (Goe06).

In this paper we move from the *externalist* oriented point of view (Man06), according to which subjective experience supposes a processual unity between the activity in the brain and the perceived event in the external world. According to externalism, the separation between subject and object must be reconsidered so that the two, while maintaining their identities as different perspectives on a process, actually occur as a unity during perception.

Starting from these considerations, we developed robots aiming to exploit sensorimotor contingencies and externalist inspired frameworks (Che07). An interesting architectural feature is the implementation of a generalized perception loop based on the perceptual space as a whole. In other words, in classic feedback only a few parameters are used to control robot behavior (position, speed, etc.). The idea behind our robots is to match a global prediction of the future perceptual state (for instance by a rendering of the visual image) with the incoming data. The goal is thus to achieve a tight coupling between robot and environment. According to these model and implementations, the physical correlate of robot subjective experience would not lie in the images internally generated but rather in the causal processes engaged between the robot and the environment.

We developed a software design methodology for implementing the generalized perception loop in an AGI system. The PASSI agent oriented methodology (Cos05) has been taken into account as the starting point (CCS09) for this new design process. Until now PASSI has been used, and proved to be useful in that, for engineering robotic applications. In this work the metaphor of *test* is employed thus providing means for designing the elements of the perception loop. We reused, modified and integrated two process fragments coming from the Unified Process (UP) *Test Plan and Design* and *Test Execution*, the former's aim is to identify the system functionalities to be tested, the available system resources and the test objective. The latter aims at executing test in order to identify defects and analyze

the results.

The new PASSI allows us to design a system that, once detected the differences between expected and real behaviors (for instance during the execution of a mission), is able to autonomously tune its parameters and learn for later generalizing the knowledge to novel situations.

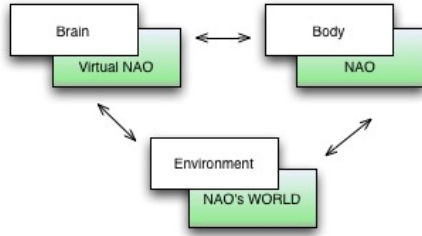


Figure 1: Realizing the Loop Brain-Body-Environment

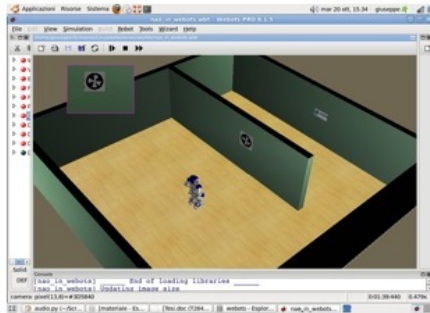


Figure 2: NAO and Virtual NAO Performing the Mission

The experiments we made aim at verifying the usability of the perception loop and the proposed software design methodology supporting that. Our aim is to create a software system able to control a robot by means of perception loops.

The robot is not equipped with “pre-compiled” pre-planning abilities: we want to study the situation (and to design that) in which the robot does not know what to do in a given case, and so it queries his memory in search of a ready solution or it tries to find a novel

solution exploiting his knowledge about itself, its capabilities and the surrounding world.

In particular, we experimented a humanoid robot NAO, developed by Aldebaran Robotics¹, endowed with a set of primitive behaviours. In order to implement the anticipation step of the NAO perception loop, we adopted the 3D robot simulator Webots from Cyberbotics². By means of Webots, we may edit NAO’s movements and behaviours as well as its surrounding environment.

Figures 1 and 2 show the design and implementation of the perception loop in NAO. We exploited the fact that we use NAO and at the same time the NAO simulator, so the perception loop among brain, body and environment (Roc05) corresponds to the loop among NAO (the real robot), the virtual NAO (the robot simulator) and the NAO’s world.

It is worth noting that the use we made of the simulation is quite different from the common one: in fact, it is not used for investigating and anticipating the robot behaviour in specific working condition, but instead for producing, starting from the designed behaviours, the expected results of a mission. The simulator and NAO work separately, only when a stop condition is identified the simulation results are compared with the real NAO parameters.

Acknowledgements This work has been partially supported by the EU project FP7-Humanobs.

References

- A. Chella, M. Cossentino, and V. Seidita. Towards a Methodology for Designing Artificial Conscious Robotic System. In A. Samsonovich, editor, *Proc. of AAAI Fall Symposium on Biologically Inspired Cognitive Architectures BICA '09*, Menlo Park, CA., 2009. AAAI Press.
- A. Chella. Towards robot conscious perception. In A. Chella and R. Manzotti, editors, *Artificial Consciousness*. Imprinting Academic, Exter, UK, 2007.
- A. Chella and R. Manzotti. Machine consciousness: A manifesto for robotics. *International Journal of Machine Consciousness*, 1(1):33 – 51, 2009.
- M. Cossentino. From requirements to code with the PASSI methodology. In *Agent Oriented Methodologies*, chapter IV, pages 79–106. Idea Group Publishing, Hershey, PA, USA, June 2005.
- B. Goertzel. *The Hidden Pattern*. BrownWalker Press, Boca Raton, 2006.
- R. Manzotti. A process oriented view of conscious perception. *Journal of Consciousness Studies*, 13(6):7 – 41, 2006.
- W.T. Rockwell. *Neither brain nor ghost*. MIT Press, 2005.

¹<http://www.aldebaran-robotics.com>

²<http://www.cyberbotics.com>

GQ(λ): A general gradient algorithm for temporal-difference prediction learning with eligibility traces

Hamid Reza Maei and Richard S. Sutton

Reinforcement Learning and Artificial Intelligence Laboratory, University of Alberta, Edmonton, Canada

Abstract

A new family of gradient temporal-difference learning algorithms have recently been introduced by Sutton, Maei and others in which function approximation is much more straightforward. In this paper, we introduce the GQ(λ) algorithm which can be seen as extension of that work to a more general setting including eligibility traces and off-policy learning of temporally abstract predictions. These extensions bring us closer to the ultimate goal of this work—the development of a universal prediction learning algorithm suitable for learning experientially grounded knowledge of the world. Eligibility traces are essential to this goal because they bridge the temporal gaps in cause and effect when experience is processed at a temporally fine resolution. Temporally abstract predictions are also essential as the means for representing abstract, higher-level knowledge about courses of action, or options. GQ(λ) can be thought of as an extension of Q-learning. We extend existing convergence results for policy evaluation to this setting and carry out a forward-view/backward-view analysis to derive and prove the validity of the new algorithm.

Introduction

One of the main challenges in artificial intelligence (AI) is to connect the low-level experience to high-level representations (grounded world knowledge). Low-level experience refers to rich signals received back and forth between the agent and the world. Recent theoretical developments in temporal-difference learning combined with mathematical ideas developed for temporally abstract options, known as intra-option learning, can be used to address this challenge (Sutton, 2009).

Intra-option learning (Sutton, Precup, and Singh, 1998) is seen as a potential method for temporal-abstraction in reinforcement learning. Intra-option learning is a type of off-policy learning. Off-policy learning refers to learning about a *target policy* while following another policy, known as *behavior policy*. Off-policy learning arises in Q-learning where the target policy is a greedy optimal policy while the behavior policy is exploratory. It is also needed for intra-option learning. Intra-option methods look inside options and allow AI agent to learn about multiple different options

simultaneously from a single stream of received data. *Option* refers to a temporally course of actions with a termination condition. Options are ubiquitous in our everyday life. For example, to go for hiking, we need to consider and evaluate multiple options such as transportation options to the hiking trail. Each option includes a course of primitive actions and only is excited in particular states. The main feature of intra-option learning is its ability to predict the consequences of each option policy without executing it while data is received from a different policy.

Temporal difference (TD) methods in reinforcement learning are considered as powerful techniques for prediction problems. In this paper, we consider predictions always in the form of answers to the questions. Questions are like “If I follow this trail, would I see a creek?” The answers to such questions are in the form of a single scalar (value function) that tells us about the expected future consequences given the current state. In general, due to the large number of states, it is not feasible to compute the exact value of each state entry. One of the key features of TD methods is their ability to generalize predictions to states that may not have visited; this is known as function approximation.

Recently, Sutton et al. (2009b) and Maei et al. (2009) introduced a new family of gradient TD methods in which function approximation is much more straightforward than conventional methods. Prior to their work, the existing classical TD algorithms (e.g.; TD(λ) and Q-learning) with function approximation could become unstable and diverge (Baird, 1995; Tsitsiklis and Van Roy, 1997).

In this paper, we extend their work to a more general setting that includes off-policy learning of temporally abstract predictions and eligibility traces. Temporally abstract predictions are essential for representing higher-level knowledge about the course of actions, or options (Sutton et al., 1998). Eligibility traces bridge between the temporal gaps when experience is processed at a temporally fine resolution.

In this paper, we introduce the GQ(λ) algorithm that can be thought of as an extension to Q-learning (Watkins and Dayan, 1989); one of the most popular off-policy learning algorithms in reinforcement learning.

Our algorithm incorporates gradient-descent ideas originally developed by Sutton et al. (2009a,b), for option conditional predictions with varying eligibility traces. We extend existing convergence results for policy evaluation to this setting and carry forward-view/backward-view analysis and prove the validity of the new algorithm.

The organization of the paper is as follows: First, we describe the problem setting and define our notations. Then we introduce the GQ(λ) algorithm and describe how to use it. In the next sections we provide derivation of the algorithm and carry out analytical analysis on the equivalence of TD forward-view/backward-view. We finish the paper with convergence proof and conclusion section.

Notation and background

We consider the problem of policy evaluation in finite state-action Markov Decision Process (MDP). Under standard conditions, however, our results can be extended to MDPs with infinite state-action pairs. We use a standard reinforcement learning (RL) framework. In this setting, data is obtained from a continually evolving MDP with states $s_t \in \mathcal{S}$, actions $a_t \in \mathcal{A}$, and rewards $r_t \in \mathbb{R}$, for $t = 1, 2, \dots$, with each state and reward as a function of the preceding state and action. Actions are chosen according to the behavior policy b , which is assumed fixed and exciting, $b(s, a) > 0, \forall s, a$. We consider the transition probabilities between state-action pairs, and for simplicity we assume there is a finite number N of state-action pairs.

Suppose the agent find itself at time t in a state-action pair s_t, a_t . The agent likes its answer at that time to tell something about the future sequence $s_{t+1}, a_{t+1}, \dots, s_{t+k}$ if actions from $t+1$ on were taken according to the option until it terminated at time $t+k$. The option policy is denoted $\pi : \mathcal{S} \times \mathcal{A} \rightarrow [0, 1]$ and whose termination condition is denoted $\beta : \mathcal{S} \rightarrow [0, 1]$.

The answer is always in the form of a single number, and of course we have to be more specific about what we are trying to predict. There are two common cases: 1) we are trying to predict the outcome of the option; we want to know about the expected value of some function of the state at the time the option terminates. We call this function the *outcome target function*, and denote it $z : \mathcal{S} \rightarrow \mathbb{R}$, 2) we are trying to predict the transient; that is, what happens during the option rather than its end. The most common thing to predict about the transient is the total or discounted reward during the option. We denote the reward function $r : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$. Finally, the answer could conceivably be a mixture of both a transient *and* an outcome. Here we will present the algorithm for answering questions with both an outcome part z and a transient part r , with the two added together. In the common place where one wants only one of the two, the other is set to zero.

Now we can start to state the goal of learning more precisely. In particular, we would like our answer to be equal to the expected value of the outcome target

function at termination plus the cumulative sum of the transient reward function along the way:

$$\begin{aligned} Q^\pi(s_t, a_t) \\ \equiv \mathbb{E}[r_{t+1} + \gamma r_{t+2} + \dots + \gamma^{k-1} r_{t+k} + z_{t+k} \mid \pi, \beta], \end{aligned} \quad (1)$$

where $\gamma \in (0, 1]$ is discount factor and $Q^\pi(s, a)$ denotes action value function that evaluates policy π given state-action pair s, a . To simplify the notation, from now on, we drop the superscript π on action values.

In many problems the number of state-action pairs is large and therefore it is not feasible to compute the action values for each state-action entry. Therefore, we need to approximate the action values through generalization techniques. Here, we use linear function approximation; that is, the answer to a question is always formed linearly as $Q_\theta(s, a) = \theta^\top \phi(s, a) \approx Q(s, a)$ for all $s \in \mathcal{S}$ and $a \in \mathcal{A}$, where $\theta \in \mathbb{R}^n$ is a learned weight vector and $\phi(s, a) \in \mathbb{R}^n$ indicates a state-action feature vector. The goal is to learn parameter vector θ through a learning method such as TD learning.

The above (1) describes the target in a Monte Carlo sense, but of course we want to include the possibility of temporal-difference learning; one of the widely used techniques in reinforcement learning. To do this, we provide an eligibility-trace function $\lambda : \mathcal{S} \rightarrow [0, 1]$ as described in Sutton and Barto (1998). We let eligibility-trace function, λ , to vary over different states.

In the next section, first we introduce GQ(λ); a general temporal-difference learning algorithm that is stable under off-policy training, and show how to use it. Then in later sections we provide the derivation of the algorithm and convergence proof.

The GQ(λ) algorithm

In this section we introduce the GQ(λ) algorithm for off-policy learning about the outcomes and transients of options, in other words, intra-option GQ(λ) for learning the answer to a question chosen from a wide (possibly universal) class of option-conditional predictive questions.

To specify the question one provides four functions: π and β , for the option, and z and r , for the target functions. To specify how the answers will be formed one provides their functional form (here in linear form), the feature vectors $\phi(s, a)$ for all state-action pairs, and the eligibility-trace function λ . The discount factor γ can be taken to be 1, and thus ignored; the same effect as discounting can be achieved through the choice of β .

Now, we specify the GQ(λ) algorithm as follows: The weight vector $\theta \in \mathbb{R}^n$ is initialized arbitrarily. The secondary weight vector $w \in \mathbb{R}^n$ is initialized to zero. An auxiliary memory vector known as the eligibility trace $e \in \mathbb{R}^n$ is also initialized to zero. Their update rules are

$$\theta_{t+1} = \theta_t + \alpha_{\theta,t} [\delta_t e_t - \kappa_{t+1} (w_t^\top e_t) \bar{\phi}_{t+1}], \quad (2)$$

$$w_{t+1} = w_t + \alpha_{w,t} [\delta_t e_t - (w_t^\top \phi_t) \phi_t], \quad (3)$$

and

$$e_t = \phi_t + (1 - \beta_t)\lambda_t \rho_t e_{t-1}, \quad (4)$$

where,

$$\delta_t = r_{t+1} + \beta_{t+1}z_{t+1} + (1 - \beta_{t+1})\theta_t^\top \bar{\phi}_{t+1} - \theta_t^\top \phi_t, \quad (5)$$

$$\bar{\phi}_t = \sum_a \pi(s_t, a) \phi(s_t, a),$$

$$\rho_t = \frac{\pi(s_t, a_t)}{b(s_t, a_t)}, \quad \kappa_t = (1 - \beta_t)(1 - \lambda_t),$$

ϕ_t is an alternate notation for $\phi(s_t, a_t)$, and $\alpha_{\theta, t} > 0$, $\alpha_{w, t} > 0$, are constant or decreasing step-size parameters for θ and w weights respectively. Here, δ_t , can be seen as one-step TD error.

In the next section we introduce a Bellman error objective function and later show that the GQ(λ) algorithm follows its gradient-descent direction and eventually converges to what that can be described as the fixed-point of TD(λ) under off-policy training.

Objective function

The key element in this paper is to extend the mean-square *projected* Bellman error objective function (MSPBE), first introduced by Sutton et al. (2009b), to the case where it incorporates eligibility traces and option-conditional probabilities. We start with an off-policy, λ -weighted version of the projected-Bellman-error objective function:

$$J(\theta) = \|Q_\theta - \Pi T_\pi^{\lambda\beta} Q_\theta\|_D^2 \quad (6)$$

where $Q_\theta = \Phi\theta \in \mathbb{R}^N$ is the vector of approximate action values for each state-action pair, Φ is an $N \times n$ matrix whose rows are the state-action feature vectors $\phi(s, a)$, $\Pi = \Phi(\Phi^\top D \Phi)^{-1} \Phi^\top D$ is a projection matrix that projects any point in the action value space into the linear space of approximate action values, D is an $N \times N$ diagonal matrix whose diagonal entries correspond to the frequency with which each state-action pair is visited under the behavior policy, $T_\pi^{\lambda\beta}$ is a λ -weighted state-action version of the affine $N \times N$ Bellman operator for the target policy π with termination probability β , and finally the norm, $\|v\|_D^2$, is defined as $v^\top D v$. The operator $T_\pi^{\lambda\beta}$ takes as input an arbitrary vector $Q \in \mathbb{R}^N$ and returns a vector giving for each state-action pair the expected corrected λ -return if the Markov decision process was started in that state-action pair, actions were taken according to π , and Q was used to correct the return truncations. When Q_θ is used for the corrections we can write

$$T_\pi^{\lambda\beta} Q_\theta(s, a) = \mathbb{E}_\pi[g_t^{\lambda\beta} | s_t = s, a_t = a], \quad (7)$$

where $g_t^{\lambda\beta}$ is the λ -return (while following behavior policy) starting from state-action pair s_t, a_t :

$$g_t^{\lambda\beta} = r_{t+1} + \beta_{t+1}z_{t+1} + (1 - \beta_{t+1})[(1 - \lambda_{t+1})\theta^\top \phi_{t+1} + \lambda_{t+1}g_{t+1}^{\lambda\beta}], \quad (8)$$

where ϕ_t is an alternate notation for $\phi(s_t, a_t)$.

It would be easier to work with this objective function if we write it in terms of statistical expectations. To do this, first, let's consider the following identities:

$$\mathbb{E}_\pi[\delta_t^{\lambda\beta} | s_t = s, a_t = a] = T_\pi^{\lambda\beta} Q_\theta(s, a) - Q_\theta(s, a),$$

where

$$\delta_t^{\lambda\beta} \equiv g_t^{\lambda\beta} - \theta^\top \phi_t, \quad (9)$$

$$\begin{aligned} \mathbb{E}_\pi[\delta_t^{\lambda\beta} \phi_t] &= \sum_{s,a} D_{sa,sa} \phi(s, a) \mathbb{E}_\pi[\delta_t^{\lambda\beta} | s_t = s, a_t = a] \\ &= \Phi^\top D (T_\pi^{\lambda\beta} Q_\theta - Q_\theta), \end{aligned}$$

and

$$\mathbb{E}_b[\phi \phi^\top] = \sum_{s,a} D_{sa,sa} \phi(s, a) \phi^\top(s, a) = \Phi^\top D \Phi.$$

Note that $D_{sa,sa}$ indicates the diagonal entry of matrix D and corresponds to the frequency with which state-action pair s, a , is visited under the behavior policy b . Here, $\mathbb{E}_\pi[\cdot] = \sum_{s,a} D_{sa,sa} \mathbb{E}_\pi[\cdot | s, a]$ because the data has been generated and observed according to the behavior policy.

Given identities above, one can follow similar steps as in Sutton et al. (2009); as follows, to show that the objective function can be written in terms of statistical expectations:

$$\begin{aligned} J(\theta) &= \|Q_\theta - \Pi T_\pi^{\lambda\beta} Q_\theta\|_D^2 \\ &= \|\Pi(T_\pi^{\lambda\beta} Q_\theta - Q_\theta)\|_D^2 \\ &= (\Pi(T_\pi^{\lambda\beta} Q_\theta - Q_\theta))^\top D (\Pi(T_\pi^{\lambda\beta} Q_\theta - Q_\theta)) \\ &= (T_\pi^{\lambda\beta} Q_\theta - Q_\theta)^\top \Pi^\top D \Pi (T_\pi^{\lambda\beta} Q_\theta - Q_\theta) \\ &= (T_\pi^{\lambda\beta} Q_\theta - Q_\theta)^\top D^\top \Phi (\Phi^\top D \Phi)^{-1} \Phi^\top D (T_\pi^{\lambda\beta} Q_\theta - Q_\theta) \\ &= (\Phi^\top D (T_\pi^{\lambda\beta} Q_\theta - Q_\theta))^\top (\Phi^\top D \Phi)^{-1} \Phi^\top D (T_\pi^{\lambda\beta} Q_\theta - Q_\theta) \\ &= \mathbb{E}_\pi[\delta^{\lambda\beta} \phi]^\top \mathbb{E}_b[\phi \phi^\top]^{-1} \mathbb{E}_\pi[\delta^{\lambda\beta} \phi], \end{aligned} \quad (10)$$

where we have used the identity $\Pi^\top D \Pi = D^\top \Phi (\Phi^\top D \Phi)^{-1} \Phi^\top D$.

In our off-policy setting, however, we cannot work with expectations conditional on π ; we need to convert them to expectations conditional on b (the behavior policy) which we can then directly sample from. To do this, we introduce an off-policy version of the multi-step TD error, $\delta_t^{\lambda\beta\rho}$,

$$\delta_t^{\lambda\beta\rho} \equiv g_t^{\lambda\beta\rho} - \theta_t^\top \phi_t, \quad (11)$$

where

$$g_t^{\lambda\beta\rho} \equiv r_{t+1} + \beta_{t+1}z_{t+1} + (1 - \beta_{t+1})[(1 - \lambda_{t+1})\theta^\top \bar{\phi}_{t+1} + \lambda_{t+1}\rho_{t+1}g_{t+1}^{\lambda\beta\rho}], \quad (12)$$

is off-policy λ -return and

$$\bar{\phi}_t = \sum_a \pi(s_t, a) \phi(s_t, a) \quad \text{and} \quad \rho_t = \frac{\pi(s_t, a_t)}{b(s_t, a_t)}.$$

The next theorem makes this transformation very simple.

Theorem 1. *Transforming conditional expectations. Let b and π denote the behavior and target policies respectively, and $\delta^{\lambda\beta}$, $\delta^{\lambda\beta\rho}$ are defined in equations (9, 11), then*

$$\mathbb{E}_\pi[\delta_t^{\lambda\beta} \phi_t] = \mathbb{E}_b[\delta_t^{\lambda\beta\rho} \phi_t]. \quad (13)$$

Proof. First, we show $\mathbb{E}_b[g_t^{\lambda\beta\rho} | s_t, a_t] = \mathbb{E}_\pi[g_t^{\lambda\beta} | s_t, a_t]$. To do this, let's write $g_t^{\lambda\beta\rho}$ (12) in the following compact form:

$$g_t^{\lambda\beta\rho} = \zeta_{t+1} + \kappa_{t+1} \theta^\top \bar{\phi}_{t+1} + (1 - \beta_{t+1}) \lambda_{t+1} \rho_{t+1} g_{t+1}^{\lambda\beta\rho},$$

where $\zeta_t \equiv r_t + \beta_t z_t$, and $\kappa_t \equiv (1 - \beta_t)(1 - \lambda_t)$. Now consider the identity $\mathbb{E}_b[\bar{\phi}_t | s_t, a_t] = \mathbb{E}_\pi[\phi_t | s_t, a_t]$ as we expand the term $\mathbb{E}_b[g_t^{\lambda\beta\rho} | s_t, a_t]$, thus we have

$$\begin{aligned} \mathbb{E}_b[g_t^{\lambda\beta\rho} | s_t, a_t] &= \mathbb{E}_b[\zeta_{t+1} + \kappa_{t+1} \theta^\top \bar{\phi}_{t+1} | s_t, a_t] \\ &\quad + \mathbb{E}_b[(1 - \beta_{t+1}) \lambda_{t+1} \rho_{t+1} g_{t+1}^{\lambda\beta\rho} | s_t, a_t] \\ &= \mathbb{E}_\pi[\zeta_{t+1} + \kappa_{t+1} \theta^\top \phi_{t+1} | s_t, a_t] \\ &\quad + \mathbb{E}_b[(1 - \beta_{t+1}) \lambda_{t+1} \rho_{t+1} g_{t+1}^{\lambda\beta\rho} | s_t, a_t] \\ &= \mathbb{E}_\pi[\zeta_{t+1} + \kappa_{t+1} \theta^\top \phi_{t+1} | s_t, a_t] \\ &\quad + \sum_{s_{t+1}} \mathbb{P}(s_{t+1} | s_t, a_t) \sum_{a_{t+1}} b(s_{t+1}, a_{t+1}) \frac{\pi(s_{t+1}, a_{t+1})}{b(s_{t+1}, a_{t+1})} \\ &\quad \times (1 - \beta_{t+1}) \lambda_{t+1} \mathbb{E}_b[g_{t+1}^{\lambda\beta\rho} | s_{t+1}, a_{t+1}] \\ &= \mathbb{E}_\pi[\zeta_{t+1} + \kappa_{t+1} \theta^\top \phi_{t+1} | s_t, a_t] \\ &\quad + \sum_{s_{t+1}} \mathbb{P}(s_{t+1} | s_t, a_t) \sum_{a_{t+1}} \pi(s_{t+1}, a_{t+1}) \\ &\quad \times (1 - \beta_{t+1}) \lambda_{t+1} \mathbb{E}_b[g_{t+1}^{\lambda\beta\rho} | s_{t+1}, a_{t+1}] \\ &= \mathbb{E}_\pi[\zeta_{t+1} + \kappa_{t+1} \theta^\top \phi_{t+1} | s_t, a_t] \\ &\quad + \mathbb{E}_\pi[(1 - \beta_{t+1}) \lambda_{t+1} \mathbb{E}_b[g_{t+1}^{\lambda\beta\rho} | s_{t+1}, a_{t+1}] | s_t, a_t], \end{aligned}$$

which as it continues to roll out, and as a result, gives us $\mathbb{E}_b[g_t^{\lambda\beta\rho} | s_t, a_t] = \mathbb{E}_\pi[g_t^{\lambda\beta} | s_t, a_t]$. From definitions of $\delta_t^{\lambda\beta}$ and $\delta_t^{\lambda\beta\rho}$, it is immediate that $\mathbb{E}_\pi[\delta_t^{\lambda\beta} \phi_t] = \mathbb{E}_b[\delta_t^{\lambda\beta\rho} \phi_t]$. \square

Thus, the objective function $J(\theta)$ in Equation (10) can be written in the following form

$$J(\theta) = \mathbb{E}_b[\delta^{\lambda\beta\rho} \phi]^\top \mathbb{E}_b[\phi \phi^\top]^{-1} \mathbb{E}_b[\delta^{\lambda\beta\rho} \phi], \quad (14)$$

in which the expectations are conditioned over behavioral policy.

Derivation of GQ(λ) algorithm: forward-view/backward-view analysis

We derive GQ(λ) algorithm based on gradient-descent in the $J(\theta)$ objective function (14). Thus, we update the modifiable parameter θ proportional to $-\frac{1}{2} \nabla J(\theta)$. Note that all the gradients in this paper are with respect to the main weight vector θ , and so are denoted simply by ∇ , thus we have

$$\begin{aligned} -\frac{1}{2} \nabla J(\theta) &= -\frac{1}{2} \nabla \left(\mathbb{E}_b[\delta^{\lambda\beta\rho} \phi]^\top \mathbb{E}_b[\phi \phi^\top]^{-1} \mathbb{E}_b[\delta^{\lambda\beta\rho} \phi] \right) \\ &= -(\nabla \mathbb{E}_b[\delta^{\lambda\beta\rho} \phi])^\top \mathbb{E}_b[\phi \phi^\top]^{-1} \mathbb{E}_b[\delta^{\lambda\beta\rho} \phi] \\ &= -\nabla \mathbb{E}_b[\delta^{\lambda\beta\rho} \phi^\top] \mathbb{E}_b[\phi \phi^\top]^{-1} \mathbb{E}_b[\delta^{\lambda\beta\rho} \phi] \\ &= -\mathbb{E}_b[(\nabla \delta^{\lambda\beta\rho}) \phi^\top] \mathbb{E}_b[\phi \phi^\top]^{-1} \mathbb{E}_b[\delta^{\lambda\beta\rho} \phi] \\ &= -\mathbb{E}_b[(\nabla g^{\lambda\beta\rho} - \phi) \phi^\top] \mathbb{E}_b[\phi \phi^\top]^{-1} \mathbb{E}_b[\delta^{\lambda\beta\rho} \phi] \\ &= (\mathbb{E}_b[\phi \phi^\top] - \mathbb{E}_b[\nabla g^{\lambda\beta\rho} \phi^\top]) \mathbb{E}_b[\phi \phi^\top]^{-1} \mathbb{E}_b[\delta^{\lambda\beta\rho} \phi] \\ &= \mathbb{E}_b[\delta^{\lambda\beta\rho} \phi] - \mathbb{E}_b[\nabla g^{\lambda\beta\rho} \phi^\top] \mathbb{E}_b[\phi \phi^\top]^{-1} \mathbb{E}_b[\delta^{\lambda\beta\rho} \phi] \\ &\approx \mathbb{E}_b[\delta^{\lambda\beta\rho} \phi] - \mathbb{E}_b[\nabla g^{\lambda\beta\rho} \phi^\top] w, \end{aligned} \quad (15)$$

where, in the final expression, we assume that we have a quasi-stationary estimate $w \in \mathbb{R}^n$ such that

$$w \approx \mathbb{E}_b[\phi \phi^\top]^{-1} \mathbb{E}_b[\delta^{\lambda\beta\rho} \phi]. \quad (16)$$

Because the expectations in the final expression (15) are not known, to update the modifiable parameter θ , we use stochastic gradient-descent approach; that is, we sample from the final expression (15) and update θ along this sample direction, where it yields the following forward-view algorithm:

$$\theta_{t+1} = \theta_t + \alpha_{\theta,t} \left(\delta_t^{\lambda\beta\rho} \phi_t - \nabla g_t^{\lambda\beta\rho} \phi_t^\top w_t \right), \quad (17)$$

where $\alpha_{\theta,t}$ is a sequence of positive step-size parameters. The desired approximation for w (16), is the solution to a least-squares problem, which can be found incrementally with linear complexity by the LMS algorithm that uses $\delta_t^{\lambda\rho}$ as its target. The standard algorithm for doing this is the following forward-view algorithm

$$w_{t+1} = w_t + \alpha_{w,t} \left(\delta_t^{\lambda\beta\rho} - w_t^\top \phi_t \right) \phi_t, \quad (18)$$

where $\alpha_{w,t}$ is another sequence of positive step-size parameters. Note that w fixed-point in the above expression is $\mathbb{E}_b[\phi \phi^\top]^{-1} \mathbb{E}_b[\delta^{\lambda\beta\rho} \phi]$.

We now turn to converting these forward-view algorithms to backward-view forms that are more convenient for low-memory mechanistic implementation. For the first term in equation (17); that is, $\delta_t^{\lambda\beta\rho}\phi_t$, which is called forward-view version of TD update, we can substitute $\delta_t e_t$ (backward-view TD update), just as in conventional TD(λ) algorithm (Sutton and Barto, 1998). This has been shown in the following theorem:

Theorem 2. *Equivalence of TD forward-view and backward-view. The forward-view description of TD update is equivalent to the mechanistic backward-view; that is,*

$$\mathbb{E}_b[\delta_t^{\lambda\beta\rho}\phi_t] = \mathbb{E}_b[\delta_t e_t], \quad (19)$$

where $\delta_t^{\lambda\beta\rho}$ is multi-step TD error, δ_t is one-step TD error and e_t denotes eligibility trace defined in equations (11, 4, 5) respectively.

Proof. We start by finding a recursive way of writing the multi-step off-policy TD error. Let $\zeta_t = r_t + \beta_t z_t$, then

$$\begin{aligned} \delta_t^{\lambda\beta\rho} &= g_t^{\lambda\beta\rho} - \theta_t^\top \phi_t \\ &= \zeta_{t+1} + (1 - \beta_{t+1}) \left[(1 - \lambda_{t+1}) \theta_t^\top \bar{\phi}_{t+1} + \lambda_{t+1} \rho_{t+1} g_{t+1}^{\lambda\rho} \right] - \theta_t^\top \phi_t \\ &= \zeta_{t+1} + (1 - \beta_{t+1}) \theta_t^\top \bar{\phi}_{t+1} - \theta_t^\top \phi_t \\ &\quad - (1 - \beta_{t+1}) \lambda_{t+1} \theta_t^\top \bar{\phi}_{t+1} + (1 - \beta_{t+1}) \lambda_{t+1} \rho_{t+1} g_{t+1}^{\lambda\beta\rho} \\ &= \delta_t \\ &\quad - (1 - \beta_{t+1}) \lambda_{t+1} \theta_t^\top \bar{\phi}_{t+1} + (1 - \beta_{t+1}) \lambda_{t+1} \rho_{t+1} g_{t+1}^{\lambda\beta\rho} \\ &\quad + (1 - \beta_{t+1}) \lambda_{t+1} (-\rho_{t+1} \theta_t^\top \phi_{t+1} + \rho_{t+1} \theta_t^\top \phi_{t+1}) \\ &= \delta_t + (1 - \beta_{t+1}) \lambda_{t+1} \rho_{t+1} (g_{t+1}^{\lambda\beta\rho} - \theta_t^\top \phi_{t+1}) \\ &\quad + (1 - \beta_{t+1}) \lambda_{t+1} \theta_t^\top (\rho_{t+1} \phi_{t+1} - \bar{\phi}_{t+1}) \\ &= \delta_t + (1 - \beta_{t+1}) \lambda_{t+1} \rho_{t+1} \delta_{t+1}^{\lambda\beta\rho} \\ &\quad + (1 - \beta_{t+1}) \lambda_{t+1} \theta_t^\top (\rho_{t+1} \phi_{t+1} - \bar{\phi}_{t+1}). \end{aligned}$$

Note that the last part of the above equation has expected value of vector zero under the behavior policy because

$$\begin{aligned} \mathbb{E}_b[\rho_t \phi_t | s_t] &= \sum_a b(s_t, a) \frac{\pi(s_t, a)}{b(s_t, a)} \phi(s_t, a) \\ &= \sum_a \pi(s_t, a) \phi(s_t, a) \equiv \bar{\phi}_t. \end{aligned}$$

Putting all these together, we can write the TD update (in expectation) in a simple way in terms of eligibility

traces which leads to backward-view:

$$\begin{aligned} \mathbb{E}_b[\delta_t^{\lambda\beta\rho}\phi_t] &= \mathbb{E}_b \left[\left(\delta_t + (1 - \beta_{t+1}) \lambda_{t+1} \rho_{t+1} \delta_{t+1}^{\lambda\beta\rho} \right) \phi_t \right] \\ &\quad + \mathbb{E}_b \left[(1 - \beta_{t+1}) \lambda_{t+1} \theta^\top (\rho_{t+1} \phi_{t+1} - \bar{\phi}_{t+1}) \phi_t \right] \\ &= \mathbb{E}_b[\delta_t \phi_t] + \mathbb{E}_b \left[(1 - \beta_{t+1}) \lambda_{t+1} \rho_{t+1} \delta_{t+1}^{\lambda\beta\rho} \phi_t \right] + 0 \\ &= \mathbb{E}_b[\delta_t \phi_t] + \mathbb{E}_b \left[(1 - \beta_t) \lambda_t \rho_t \delta_t^{\lambda\beta\rho} \phi_{t-1} \right] \\ &= \mathbb{E}_b[\delta_t \phi_t] + \mathbb{E}_b \left[(1 - \beta_t) \lambda_t \rho_t \left(\delta_t + (1 - \beta_{t+1}) \lambda_{t+1} \rho_{t+1} \right. \right. \\ &\quad \left. \left. \times \delta_{t+1}^{\lambda\beta\rho} + (1 - \beta_{t+1}) \lambda_{t+1} \theta^\top (\rho_{t+1} \phi_{t+1} - \bar{\phi}_{t+1}) \right) \phi_{t-1} \right] \\ &= \mathbb{E}_b[\delta_t \phi_t] + \mathbb{E}_b \left[(1 - \beta_t) \lambda_t \rho_t \delta_t \phi_{t-1} \right] \\ &\quad + \mathbb{E}_b \left[(1 - \beta_t) \lambda_t \rho_t (1 - \beta_{t+1}) \lambda_{t+1} \rho_{t+1} \delta_{t+1}^{\lambda\beta\rho} \phi_{t-1} \right] + 0 \\ &= \mathbb{E}_b[\delta_t (\phi_t + (1 - \beta_t) \lambda_t \rho_t \phi_{t-1})] \\ &\quad + \mathbb{E}_b \left[(1 - \beta_{t-1}) \lambda_{t-1} \rho_{t-1} (1 - \beta_t) \lambda_t \rho_t \delta_t^{\lambda\beta\rho} \phi_{t-2} \right] \\ &\quad \vdots \\ &= \mathbb{E}_b \left[\delta_t \left(\phi_t + (1 - \beta_t) \lambda_t \rho_t \phi_{t-1} \right. \right. \\ &\quad \left. \left. + (1 - \beta_t) \lambda_t \rho_t (1 - \beta_{t-1}) \lambda_{t-1} \rho_{t-1} \phi_{t-2} + \dots \right) \right] \\ &= \mathbb{E}_b[\delta_t e_t], \end{aligned} \quad (20)$$

where $e_t = \phi_t + (1 - \beta_t) \lambda_t \rho_t e_{t-1}$, which gives us a backward view algorithm for the TD(λ) update. \square

For the second term of the gradient update (15), we can use the following trick: we take the gradient of the forward-backward relationship just established in theorem 2; that is, $\nabla \mathbb{E}_b[\delta_t^{\lambda\beta\rho}\phi_t] = \nabla \mathbb{E}_b[\delta_t e_t]$, then $\mathbb{E}_b[\nabla \delta_t^{\lambda\beta\rho} \phi_t^\top] = \mathbb{E}_b[\nabla \delta_t e_t^\top]$, and consequently we get, $\mathbb{E}_b[\nabla g_t^{\lambda\beta\rho} \phi_t^\top] - \mathbb{E}_b[\phi_t \phi_t^\top] = \mathbb{E}_b[\nabla ((1 - \beta_{t+1}) \bar{\phi}_{t+1} - \phi_t) e_t^\top]$. By arranging the terms and using Equation (4), and $\mathbb{E}_b[\rho_t \phi_t | s_t = s] = \bar{\phi}_t$, we get

$$\begin{aligned} \mathbb{E}_b[\nabla g_t^{\lambda\beta\rho} \phi_t^\top] &= \mathbb{E}_b[\phi_t \phi_t^\top] + \mathbb{E}_b[(1 - \beta_{t+1}) \bar{\phi}_{t+1} e_t^\top] - \mathbb{E}_b[\phi_t e_t^\top] \\ &= \mathbb{E}_b[\phi_t \phi_t^\top] + \mathbb{E}_b[(1 - \beta_{t+1}) \bar{\phi}_{t+1} e_t^\top] \\ &\quad - \mathbb{E}_b[\phi_t (\phi_t + (1 - \beta_t) \lambda_t \rho_t e_{t-1})^\top] \\ &= \mathbb{E}_b[(1 - \beta_{t+1}) \bar{\phi}_{t+1} e_t^\top] - \mathbb{E}_b[(1 - \beta_t) \lambda_t \rho_t \phi_t e_{t-1}^\top] \\ &= \mathbb{E}_b[(1 - \beta_{t+1}) \bar{\phi}_{t+1} e_t^\top] - \mathbb{E}_b[(1 - \beta_t) \lambda_t \bar{\phi}_t e_{t-1}^\top] \\ &= \mathbb{E}_b[(1 - \beta_{t+1}) \bar{\phi}_{t+1} e_t^\top] - \mathbb{E}_b[(1 - \beta_{t+1}) \lambda_{t+1} \bar{\phi}_{t+1} e_t^\top] \\ &= \mathbb{E}_b[(1 - \beta_{t+1}) (1 - \lambda_{t+1}) \bar{\phi}_{t+1} e_t^\top]. \end{aligned} \quad (21)$$

Returning now to the forward-view equation for updating θ (17), it should be clear that for the first term

we can substitute $\delta_t e_t$, based on (19), just as in conventional TD(λ), and for the second term we can substitute based on (21), thus the backward-view update is as follows:

$$\theta_{t+1} = \theta_t + \alpha_{\theta,t} [\delta_t e_t - \kappa_{t+1} (e_t^\top w_t) \bar{\phi}_{t+1}], \quad (22)$$

where $\kappa_t = (1 - \beta_t)(1 - \lambda_t)$. The forward-view algorithm for w , (18), is particularly simple to convert to a backward-view form. The first term is again the same as the conventional linear TD(λ) update, and the second term is already in a suitable mechanistic form. The simplest backward-view update is

$$w_{t+1} = w_t + \alpha_{w,t} [\delta_t e_t - (w_t^\top \phi_t) \phi_t]. \quad (23)$$

Convergence of GQ(λ)

In this section, we show that GQ(λ) converges with probability one to the TD(λ) fixed-point under standard assumptions. The TD(λ) fixed-point, θ^* , is a point which satisfies in

$$0 = \mathbb{E}_b[\delta_t e_t] = -A\theta^* + b, \quad (24)$$

where

$$A = \mathbb{E}_b[e_t (\phi_t - (1 - \beta_{t+1})\bar{\phi}_{t+1})]^\top, \quad (25)$$

$$b = \mathbb{E}_b[(r_{t+1} + \beta_{t+1}z_{t+1}) e_t]. \quad (26)$$

Theorem 3. *Convergence of GQ(λ). Consider the GQ(λ) iterations (2,3,4) with step-size sequences $\alpha_{\theta,t}$ and $\alpha_{w,t}$ satisfying $\alpha_{\theta,t}, \alpha_{w,t} > 0$, $\sum_{t=0}^{\infty} \alpha_{\theta,t} = \sum_{t=0}^{\infty} \alpha_{w,t} = \infty$, $\sum_{t=0}^{\infty} \alpha_{\theta,t}^2 < \infty$ and $\sum_{t=0}^{\infty} \alpha_{w,t}^2 < \infty$ and that $\frac{\alpha_{\theta,t}}{\alpha_{w,t}} \rightarrow 0$ as $t \rightarrow \infty$. Further assume that ϕ_t is a Markov process with a unique invariant distribution and that the ϕ_t , e_t , z_t , and r_t sequences have uniformly bounded second moments. Assume that A (25) and $C = \mathbb{E}_b[\phi_t \phi_t^\top]$ are non-singular matrices. Then the parameter vector θ_t converges with probability one to the TD(λ) fixed-point θ^* (24).*

Proof. We use Lemma 6.7 (Bertsekas and Tsitsiklis 1996) that can be applied here and follow the proof of convergence for the TDC algorithm in Sutton et al. (2009b). For the brevity, we have omitted the proof. \square

Conclusion

The GQ(λ) algorithm, which has been introduced in this paper, incorporates varying eligibility traces and option-conditional probabilities for policy evaluation. To derive GQ(λ), we carried out a forward-view/backward-view analysis. We extended the existing convergence results to show that GQ(λ) is guaranteed to converge to the TD(λ) fixed-point. GQ(λ) is a general gradient TD method for off-policy learning and as such can be seen as extension of Q-learning. GQ(λ) is able to learn about temporally abstract predictions,

which makes it suitable to use for learning experientially grounded knowledge. In addition, GQ(λ) is online, incremental and its computational complexity scales only linearly with the size of features. Thus, it is suitable for large-scale applications. Our work, however, is limited to policy evaluation. Interesting future works is to extend GQ(λ) for control problems and gather extensive empirical data on large-scale real-world applications.

References

- Baird, L. C. (1995). Residual algorithms: Reinforcement learning with function approximation. In *Proceedings of the Twelfth International Conference on Machine Learning*, pp. 30–37. Morgan Kaufmann.
- Bertsekas, D. P., Tsitsiklis, J. (1996). *Neuro-Dynamic Programming*. Athena Scientific, 1996.
- Maei, H. R., Szepesvári, Cs, Bhatnagar, S., Precup, D., Silver D., Sutton, R. S. (2009). Convergent Temporal-Difference Learning with Arbitrary Smooth Function Approximation. In *Accepted in Advances in Neural Information Processing Systems 22*. MIT Press.
- Sutton, R. S. (1988). Learning to predict by the method of temporal differences. *Machine Learning* 3:9–44.
- Sutton, R. S., Barto, A. G. (1998). *Reinforcement Learning: An Introduction*. MIT Press.
- Sutton, R. S., Precup, D., Singh, S. (1998). Intra-option learning about temporally abstract actions. *Proceedings of the 15th International Conference on Machine Learning*, pp. 556–564.
- Sutton, R. S., Szepesvári, Cs., Maei, H. R. (2009a). A convergent O(n) algorithm for off-policy temporal-difference learning with linear function approximation. In *Advances in Neural Information Processing Systems 21*. MIT Press.
- Sutton, R. S., Maei, H. R., Precup, D., Bhatnagar, S., Silver, D., Szepesvári, Cs., Wiewiora, E. (2009b). Fast gradient-descent methods for temporal-difference learning with linear function approximation. In *Proceedings of the 26th International Conference on Machine Learning, Montreal, Canada*.
- Sutton, R. S. (2009). The grand challenge of predictive empirical abstract knowledge. *Working Notes of the IJCAI-09 Workshop on Grand Challenges for Reasoning from Experiences*.
- Tsitsiklis, J. N., and Van Roy, B. (1997). An analysis of temporal-difference learning with function approximation. *IEEE Transactions on Automatic Control* 42:674–690.
- Watkins, C. J. C. H., and Dayan, P. (1992). Q-learning. *Machine Learning*, 8:279–292.

Stochastic Grammar Based Incremental Machine Learning Using Scheme

Eray Özkural and Cevdet Aykanat

Bilkent University
Ankara, Turkey

Introduction

Gigamachine is our initial implementation of an Artificial General Intelligence (AGI system) in the O’Caml language with the goal of building Solomonoff’s “Phase 1 machine” that he proposed as the basis of a quite powerful incremental machine learning system (Sol02). While a lot of work remains to implement the full system, the present algorithms and implementation demonstrate the issues in building a realistic system. Thus, we report on our ongoing research to share our experience in designing such a system. In this extended abstract, we give an overview of our present implementation, summarize our contributions, discuss the results obtained, the limitations of our system, our plans to overcome those limitations, potential applications, and future work.

The reader is referred to (Sch04; Sol02; Sol09) for a background on general-purpose incremental machine learning. The precise technical details of our ongoing work may be found in (Ozk09), which focuses on our algorithmic contributions. The discussion here is not as technical but assumes basic knowledge of universal problem solvers.

An overview of Gigamachine

Gigamachine is an incremental machine learning system that works on current gigaflop/sec scale serial computers. It is the testbed for our experiments with advanced general purpose incremental machine learning. Here, we describe version 1 of Gigamachine. Incremental machine learning can act as the kernel of complete AGI systems such as Solomonoff’s Q/A machine or the Gödel Machine. The present implementation solves operator induction over example input/output pairs (any Scheme expression is an example). Our work may be viewed as an alternative to OOPS in that regard.

Design and implementation choices

Reference machine In Algorithmic Probability Theory, we need to fix a universal computer as the reference machine. (Sol09) argues that the choice of a reference machine introduces a necessary bias to the learning system and looking for the “ultimate machine” may be a red herring. In previous work, low-level universal computers such as FORTH and Solomonoff’s AZ have been proposed. Solomonoff suggests APL, LISP, FORTH, or assembly. We have chosen the

LISP-like language Scheme R5RS, because it is a high level functional language that is quite expressive and flexible, directly applicable to real-world problems. We have taken all of R5RS and its standard library with minor omissions.

Probability model of programs We use a stochastic Context-Free Grammar (CFG) as the “guiding pdf” for our system as suggested by Solomonoff (Sol09). Although Solomonoff proposes some methods to make use of the solution corpus using stochastic CFG’s, we introduce complete algorithms for both search and update.

Implementation language We have chosen O’Caml as our implementation platform, as this modern programming language makes it easier to write sophisticated programs yet works efficiently. The present implementation was completed in about a month.

Implementation platform We have used an ordinary desktop computer and serial processing to test our ideas, we will use more advanced architectures as we increase the complexity of our system.

Contributions

We have made several contributions to incremental machine learning regarding both *search* and *update* algorithms. A good part of our contributions stem from our choice of Scheme as a reference computer. It would seem that choosing Scheme also solves some problems with low-level languages. A drawback in systems like OOPS is that they do not make good use of memory, better update algorithms may eventually alleviate that drawback.

Search algorithms

For the search algorithm, we use Depth-First Search in the space of *derivations* of the stochastic CFG. Thus, only syntactically correct candidates are generated. We use left-most derivation to derive programs from the start symbol. We keep track of defined variables and definitions to avoid generating unbound references and definitions by extending CFG with procedural rules. We generate variable declarations and integers according to the Zeta distribution which has empirical support. We use a probability horizon to limit the search depth. We also propose using best-first search and a new memory-aware hybrid search method.

Update algorithms

We have designed four update algorithms that we will summarize. The former two have been implemented and they contain significant innovations on top of existing algorithms. The latter two are completely novel algorithms. All of them are well adapted to stochastic CFG's and Scheme.

Modifying production probabilities We use derivations of the programs in the solution corpus to update production probabilities. Since each derivation consists of a sequence of productions applied to nonterminals in sentential forms, we can easily compute probabilities of productions in the solution corpus. This is easy to do since the search already yields the derivations. However, this would cause zero probabilities for many productions if we used those probabilities directly, thus we use exponential smoothing to avoid zero probabilities.

Re-using previous solutions Modifying probabilities is not enough as there is only so much information that it can add to the stochastic CFG. We extend the stochastic CFG with previously discovered solutions (Scheme definitions) and generate candidates that use them. This is very natural in Scheme as the syntactic extension of a function is not a function. In principle, this is synergistic with modifying production probabilities as the probabilities of new productions can be updated, although in practice this depends on implementation details.

Learning programming idioms We can learn more than the solution itself by remembering syntactic abstractions that lead to the solution. Syntactic abstraction removes some levels of derivation to obtain abstract programs and then remembers them using the algorithm of re-using previous solutions.

Frequent sub-program mining Similar to code re-use, we can find frequently occurring sub-programs in the solution corpus as Scheme expressions and add them to the grammar.

Training Sequence and Experiments

We have tested a simple training sequence that consists of the identity, square, addition, test if zero, fourth power, NAND, NOR, XOR, and factorial functions. Details can be found in (Ozk09). Our experiments show beyond doubt the validity of our update algorithms. The search times decrease dramatically for similar subsequent problems showing the effectiveness of modifying probabilities. The fourth power function demonstrates code re-use in its solution of `(define (pow4 x) (define (sqr x) (* x x)) (sqr (sqr x)))` which takes shorter than solving the square problem itself. The factorial function took more than a day, so we interrupted it. It would be eventually found like other simple problems in the literature, however, we think that it showed us that we should improve the efficiency of our algorithms.

Discussion, Applications and Future Work

The slowness of searching the factorial function made us realize that we need improvements in both the search and the

update algorithms. Some doubts have been raised whether our system can scale up to AGI since the search space is vast. In AGI, the search space is always vast, whether is the solution space, program space, proof space, or another space. Since no system has been shown to be able to write any substantially long program, we think that these doubts are premature. The path to bootstrapping most likely lies in more sophisticated search and update/memory mechanisms for a general purpose induction machine. Therefore, we think that we should proceed by improving upon the existing system. Regarding search, we can try to avoid semantically incorrect programs and try to consider time and space complexity of candidate solutions. The approach of HSEARCH may be applied. The probability model can be further advanced. For update, ever more sophisticated algorithms are possible. Another major direction that Solomonoff has suggested is context-aware updates, which may require significant changes.

The most important promise of *initial* AGI implementations will be to decrease the human contribution in *current* AI systems. The heuristic programmers of old school AI research can be replaced by programs like the Gigamachine. General induction might act as the “glue code” that will make common sense knowledge bases and natural language processing truly work. Many problems in AI are solvable only because the researchers were clever enough to find a good representation. AGI programs may automate this task. The current programs in machine learning and data mining may be supplemented by AGI methods to yield much more powerful systems. In particular, hybrid systems may lead to more intelligent ensemble learners and general data mining.

We will develop a more realistic training sequence featuring recursive problems, optimizing search and implementing the remaining two update algorithms. After that, we will extend our implementation to work on parallel multi-core and/or GPU hardware. Those new architectures are extremely suitable for our system which will not require much synchronization between cores and requires little memory per core. We will then complete the implementation of Phase 1, implement the Phase 2 of Solomonoff's system, and attempt implementing other AGI proposals such as the Gödel Machine on top of our AGI kernel.

References

- [Ozk09] Eray Ozkural. Gigamachine: incremental machine learning on desktop computers. <http://examachine.net/papers/gigamachine-draft.pdf>, December 2009. Draft.
- [Sch04] Juergen Schmidhuber. Optimal ordered problem solver. *Machine Learning*, 54:211–256, 2004.
- [Sol02] Ray Solomonoff. Progress in incremental machine learning. In *NIPS Workshop on Universal Learning Algorithms and Optimal Search*, Whistler, B.C., Canada, December 2002.
- [Sol09] Ray Solomonoff. Algorithmic probability: Theory and applications. In M. Dehmer and F. Emmert-Streib, editors, *Information Theory and Statistical Learning, Springer Science+Business Media*, pages 1–23. N.Y., 2009.

An Artificial Intelligence Model that Combines Spatial and Temporal Perception

Jianglong Nan and Fintan Costello

School of Computer Science and Informatics

University College Dublin

Belfield, Dublin 4, Ireland

jianglong.nan@ucdconnect.ie, fintan.costello@ucd.ie

Abstract

This paper proposes a continuous-time machine learning model that learns the chronological relationships and the intervals between events, stores and organises the learnt knowledge in different levels of abstraction in a network, and makes predictions about future events. The acquired knowledge is represented in a categorisation-like manner, in which events are categorised into categories of different levels. This inherently facilitates the categorisation of static items and leads to a general approach to both spatial and temporal perception. The paper presents the approach and a demonstration showing how it works.

Introduction

The general goal of artificial intelligence requires the intelligent agent to learn and acquire the knowledge not only from the data set that represents a static environment, but also from a dynamic world in which some events may always happen after others and the intervals between them may follow some patterns. Tackling both the spatial and the temporal aspects of the problem has attracted the interest of other researchers as well (Sur09).

Intelligence arises first from getting to know the similarities and differences between different items. This has been well studied by psychologists in the area of categorisation and concept formation. However, in the very wide range of models presented in the literature of this area (Ros78) (Ham95) (Nos86) (Kru92), the items are presented as a set of features in a static way, in which each presentation is a single discrete event, independent of and unrelated to all other events.

The shortcomings of this static view of concept formation lie in two aspects. Firstly, it lacks the ability to deal with irregular dynamic events that last a period of time, which, we think, can also be handled in a similar way to the static items. And secondly, human beings and animals actually always perceive an environment that continuously changes. What is missing in the static view is how the dynamic perception can be transformed into the static items.

An experiment described in (Car02) showed that the vision of our eyes is actually very small. What is usually thought to be observed at once by the eyes is actually perceived piece by piece and step by step. To our nervous system, even the perception of a static world is a multi-step ob-

servation process that spans over a period of time, which needs to be converted to or treated as an integrated item somehow to facilitate the intelligence of a higher level.

On the other hand, the learning of time-spanning events has been studied independently of concept formation. In Pavlov's studies (Pav28), dogs were trained or conditioned by being presented with a conditioning stimulus (e.g. the ringing of a bell) that was followed, after a certain controlled interval, by an unconditioned stimulus (e.g. the presentation of food). The critical observation in these studies was that the timing of a trained dog's conditioned response (e.g. salivation) depended on the interval between the conditioned and unconditioned stimuli (between the ringing of the bell and the presentation of food). The longer the interval between the bell and the presentation of food, the longer the interval between the bell and the start of salivation: the shorter the interval between the bell and food, the shorter the interval between the bell and salivation. Skinner (Ski38) similarly found that the timing of operant response in trained pigeons was also dictated by the intervals between reinforcement.

However, unlike the studies of categorisation, all these studies focused only on one or two specific events and the corresponding intervals, rather than the relationships between various events and the acquisition of knowledge. This paper presents a machine learning approach based on the argument that time-spanning events can be considered in the same way as the concepts that are learnt and categorised in the traditional categorisation perspective.

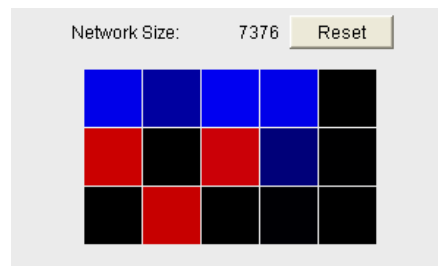


Figure 1: Demonstration of our Machine Learning Approach at <http://csserver.ucd.ie/~jlongnan/agi.html>. We invite readers to play with this demo before reading on.

An agent-environment perception is taken. And the model is designed to serve the purpose of making predictions of what will happen in the environment.

The remainder of the paper is organised as follows. First, we present an overview of the model. Then, we go through different aspects of the model in detail. Finally, we present a demonstration as illustrated in Figure 1 that shows how the model works.

Overview

Basically, the model is based on a network of knowledge. Each node of the network is a rule in the form of

If something happens, then something (else) will happen, in some time.

which includes three elements: an *antecedent*, a *consequence*, and an *interval*. Note that the antecedent and the consequence can be the same thing.

Such a node can also be considered as an individual *event* in which

First something happens, and then after some time, something (else) happens.

And this event, represented by an individual node in the network, can take the role of the antecedent or the consequence of some other rules, i.e. nodes.

In this way, nodes are connected with each other. And although a node consists of only one antecedent and one consequence, it can represent a long series if it is on a higher level and thus contains a lot of other lower level nodes indirectly. Ultimately, every node refers to a set of *sensory inputs*. Despite that we can create as many nodes as needed, we do have a fixed set of sensory inputs for any given agent in this model. These sensory inputs are supposed to receive pulse-like stimuli to perceive the changes of the environment. And the nodes not only define sets of sensory inputs, but also define the patterns and rhythms in which they get stimulated. Figure 2 shows an example. Note that the interval between the antecedent and the consequence is defined to be the interval between the first stimulus of the antecedent and the first stimulus of the consequence.

Unlike the Temporal Causal Networks described by Bouzid and Ligeza (BL00), the causality of not only the individual inputs but also specific sequences of inputs is represented in this network.

This knowledge representation is designed to allow the predictions of what will happen in the environment to be made in a distributed manner. Each node, having observed that its antecedent has happened or is happening, makes a prediction that its consequence will happen.

Both the concrete experiences and the abstractions are represented by and handled through the network nodes in exactly the same way. For a particular series of perceived stimuli, in addition to the node representing the whole event, different nodes covering different aspects of the event are also created. The latter is considered the abstraction.

Less abstract nodes refer to more abstract ones. For example, in Figure 2, A refers to B and D. A represents that if S1, S2 and S3 get stimulated in this particular pattern then

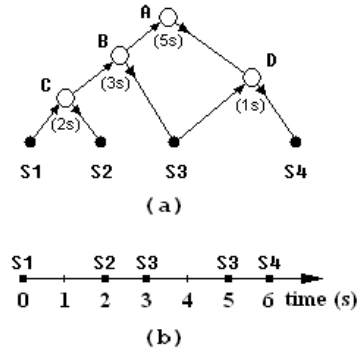


Figure 2: An Example of the Network: **(a)** Four nodes, A, B, C, and D, denoted by the circles are formed based on four sensory inputs, S1, S2, S3, and S4, which are denoted by the dots. The intervals are written in the brackets below the nodes. Arrows are used to point the antecedent of a node to the node and also the node to its consequence. The antecedent and consequence of a node are always drawn below it. **(b)** The series represented by node A happens first with S1 getting stimulated. Assuming that it is at time 0, S2 then gets stimulated at time 2s. S3 gets stimulated twice at time 3s and 5s respectively. And finally S4 gets stimulated at time 6s. As mentioned previously, the interval between B and D is defined to be the interval between the time when S1 gets stimulated and the time when S3 gets stimulated for the second time as the antecedent of D.

S3 and S4 will get stimulated in another pattern. It is considered less abstract than B and D because it contains more details. A part of A, e.g. B, may be the useful part while the rest may be just the trivial details. In this case, A can be considered as an instance of category B. A node can be referred to by multiple other nodes. This leads to a semi-hierarchical network, in which the knowledge represented by the nodes may arbitrarily overlap with one another and no strict tree structure is constructed. The nodes can be viewed not only as different events, but also as the categories they fall into. A more abstract node can be considered as the category of all the less abstract nodes that refer to it. And in this way, the network can be considered both as a collection of events and as the categorisation of these events.

This model deals with categorisation from a perspective of perception. The perception is categorised based on the similarity between different experiences.¹ For example, *my car as observed this morning* and *my car as observed this evening* are considered as two different instances or items. Both of them are represented by a node and they may both refer to a more abstract one representing *my car*. In other words, they are categorised as *my car*. Meanwhile, *my car*

¹Although we do believe the things that are categorised together by human beings must, from a perspective of perception, share something in common, e.g., hearing other people call those things the same name, this paper does not discuss whether this principle is appropriate or not. The proposed prediction-oriented model inherently deals with categorisation this way.

may refer to and thus be categorised into a more abstract node, *car*. Unlike the traditional categorisation models such as (Ros78) and (Kru92), this model does not differentiate between the items and the categories they fall into. In the above example, *my car* is not only an item but also a category.

The generalisation happens when some nodes are considered more reliable than others. The nodes that are more likely to make the right predictions and more reliable in reflecting what really is going on in the environment dominate the others.

The following sections focus on the three major aspects of the model, which are *learning*, *recognition*, and *prediction* respectively.

Learning

Event

Unlike most conventional machine learning models, the sensory system of this model is designed to receive pulse-like stimuli input instead of maintain a set of state variables that can be retrieved freely. The intelligent agent possesses an array of sensory inputs. An *event* is simply one or more sensory inputs getting stimulated in some pattern, in which an individual sensory input may be stimulated more than once.

For the sake of discussion, an event is considered to be a *sub-event* of another if all its stimuli are also presented in the other, and the intervals between them are the same. Equivalently, the latter is called a *super-event* of the former.

There are two types of sub-events. A *segment* sub-event is a continuous fragment of its super-event. It contains every stimulus in this fragment. While a *non-segment* sub-event is a concatenation of more than one continuous segments, with missing stimuli from its super-event in between. Figure 3 illustrates this.

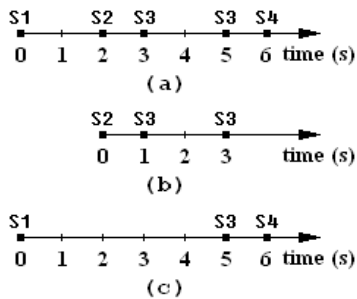


Figure 3: Segment Sub-events and Non-segment Sub-events: **(a)** The event denoted by node A in Figure 2, in which the sensory inputs S1, S2, S3 and S4 get stimulated in a specific tempo. **(b)** A segment sub-event of it. **(c)** A non-segment sub-event of it.

Memory

As described previously, network nodes are formed to denote the memories of the perceived events in the form of antecedent-consequence pairs with intervals, for future use.

Simple nodes, which usually form in the early stage of a simulation, have sensory inputs as both their antecedents and consequences. They represent simple events that consist of only two stimuli to the sensory inputs. Meanwhile, nodes that represent long and complicated events can be formed based on existing nodes.

Concrete Experience

When an event is perceived, i.e. some sensory inputs are stimulated in some pattern, the event itself and all its sub-events will be stored, assuming none of them has ever been perceived before.

Take the event shown in Figure 4 (a) for example. First, as shown in Figure 4 (b), a node for the first two stimuli is formed based directly on the sensory inputs (Node A1). Next, a node for the first three stimuli is formed based on the first node and the third sensory input (Node A2). The rest of the event stimuli is all included this way. And finally the node for the whole event is formed (Node A).

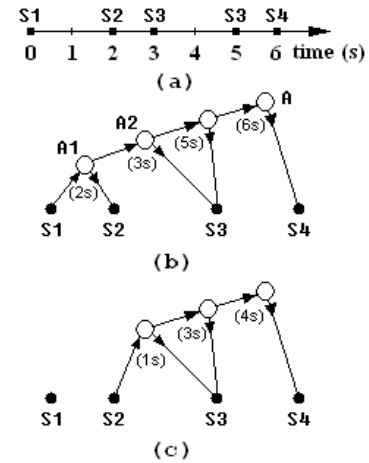


Figure 4: Nodes Formed for Segment Sub-events

But that's not all. Only the nodes for the segment sub-events that start with the first stimulus have been formed. In addition to that, the nodes for the segment sub-events starting with the second stimulus of the event are formed in a similar way as shown in Figure 4 (c). And the nodes for the segment sub-events starting with each of the rest of the stimuli are all formed this way.

As a result, a node is formed for every possible fragment of the event, which may start from anywhere in the event and end anywhere after the starting point.

Abstraction

Abstraction, in this continuous-time event-based model, is viewed as the formation of sub-events that are shared by multiple events.

Sub-events that may possibly be contained by the events perceived in the future form nodes so that they can be referred to later. This actually has been partially demonstrated in the example shown in Figure 4, in which the nodes

for only one type of the sub-events of the perceived event, namely the segment ones, are formed.

Actually, the nodes for all the non-segment sub-events are also formed similarly. For example, the second and the fourth stimulus of an event will form a node with the interval between them, based on which another node for the second, the fourth, and the fifth stimulus will also be formed.

Any sub-event of a perceived event is considered to be a possible pattern that the event may be following. When a number of events indeed follow a particular pattern, they share that sub-event by referring to the node representing it either directly as the antecedent or the consequence of them, or indirectly. Meanwhile, the node for the pattern itself, is probably sharing some more general patterns with other nodes. And a network of events of different levels of detail, that is, different levels of abstraction, is formed this way.

The generalisation of the model works in a way in which, in the beginning of the learning, the perceived item is analysed and all various aspects of it are represented individually. These aspects, which overlap with each other, actually include all possible more general and more abstract items, i.e. categories. For example, when *a particular car as observed this morning* is perceived, various aspects of it such as *a particular car*, *car*, *vehicle*, *wheel*, *a particular wheel* and a lot of things we do not have names for are also represented and stored at the same time. As more and more items are perceived by the agent, some aspects of the perception, e.g. *car*, are found to be more useful than others, e.g. *a particular wheel*, because they are shared by more items and thus are more reliable in making the predictions. These aspects gain more credit and have stronger influence. And they can be considered as the basic level categories defined by Rosch (Ros78).

Short-term Memory

The nodes that are formed for a single event are discussed in the previous sections. However, the intelligent agent is supposed to receive a continuous series of stimuli, without separators.

One option is to take all the stimuli from the start till present as a big event. But that will cost too much space and time. In the model, when a sensory input is stimulated, only a limited number of previous stimuli will be linked to it to form new nodes. The stimuli that occur too early are simply ignored, as if they have never occurred. The number of nodes that are formed is limited this way. Those stimuli that are used to form new nodes are considered to be in a *short-term memory*.

A limited size of short-term memory may seem to completely prevent long series from being learnt. But actually it does not. This is discussed in the next section.

Recognition

Recurring Event

When an event that has been previously perceived occurs, no new node is formed. Rather, the event is *recognised* by an existing node. This *recognition* process takes place when

1. The antecedent and consequence of an existing node is perceived and both of them are still being held in the short-term memory; and
2. The difference between the perceived interval and that of the existing node is within a tolerance range.

When either of the above two conditions is not met, even if the other one is, no recognition takes place and instead a new node is formed.

Familiarity

The capacity of the short-term memory is designed in relation to the number of nodes instead of the number of stimuli of the events being held.² When the recognition process happens, the existing node will take the place of its antecedent and consequence in the short-term memory. This makes more space in the short-term memory so that more information can be held in it. Note that just like more than one stimulus of a single sensory input can be held in the short-term memory, a single node can be recognised more than once in a short period of time and thus has more than one instance being held in the short-term memory.

Recognition can happen recursively. The nodes that refer to other nodes can be recognised after their antecedents and consequences are recognised.

The learning process can also take place upon the recognition. New nodes can be formed based on the recognised ones or combinations of the recognised nodes and raw stimuli.

Therefore, no matter how long a series is, it can still be learnt piece by piece, as long as it recurs enough times.

Overlapping Events

Assuming two events that have already been learnt independently recur in an overlapping manner, in which, one event starts before the other finishes. Both events will be recognised. And a new node will be formed with one of them as the antecedent and the other as the consequence. The interval, by definition, is the delay between their first stimuli.

Such nodes, with their antecedents overlapping with their consequences, are dealt with in the same way as other nodes. They are also used to make the predictions, as discussed in detail in the next section.

Prediction

Knowledge and its Use

A node is not only an event. It is also a piece of *knowledge*. It is a fraction of the understanding of what the environment is like. It is a prediction that if its antecedent happens, its consequence happens after the interval.

The possession of a collection of nodes is the possession of the knowledge of some aspects of the environment. To make use of the knowledge, an intelligent agent makes *predictions*.

²In the current model, the short-term memory is represented by a fixed number of spaces, each of which can store either a stimulus or a recognised node.

The sole purpose of this knowledge structure is to allow the model to *make predictions of what will happen based on what has already happened*. Whenever the antecedent of a node is perceived, it makes a prediction that its consequence will happen after the interval. A large number of predictions may be made simultaneously by multiple nodes from all over the network. These predictions can then be used to make decisions when the model is put into a decision making process.

Confidence

The prediction is actually made in the form of the *confidence*, which is an attribute of the node, ranging from 0 to 1. From one perspective, the confidence of a node can be viewed as the degree of certainty held by the node that its consequence is about to happen at the given time. From another perspective, it can also be viewed as the degree of certainty held by the node that the event represented by the node itself is actually happening at the given time.

The confidence of a node is determined by various factors, the most important of which is the time since its antecedent last happened.

We define the *expectation* of a node, which ranges from 0 to 1. It usually stays at 0. When the antecedent of the node is perceived and the exact interval has passed, it reaches 1. The time when the expectation reaches 1 is called the *expected time*. When the time is around the expected time and within a tolerance range, the expectation gets a value between 0 and 1. The closer the time is to the expected time, the higher the expectation is. Currently a cosine function is adopted. See Figure 5 for illustration.

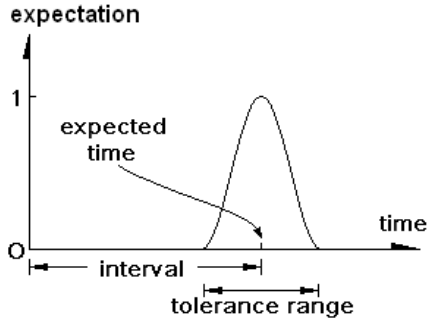


Figure 5: Expectation Changing over Time: Assuming the antecedent of the node occurs at time 0 and the consequence does not happen, the expectation of the node starts to increase when the time is within the tolerance range and reaches its maximum value of 1 at the expected time. After that, it gradually decreases to 0.

If the consequence of the node does happen when its expectation is greater than 0, the prediction is considered to be verified and the node stops making the prediction by changing its expectation to 0 immediately.³

³Note that the antecedent is assumed to have completely happened at this point. More complicated cases are discussed in subsequent sections.

In the simplest case, the confidence of a node is simply its expectation as in

$$c = e \quad (1)$$

where c is the confidence of the node and e is the expectation of it. Other factors that influence the confidence are discussed in subsequent sections.

Reliability

The predictions made by different nodes are not treated the same way.

A prediction made by a particular node at a specific time may turn out to be either right or wrong. On the one hand, some nodes may tend to always make right predictions. On the other hand, some nodes may tend to always make wrong predictions. The knowledge represented by the former better reflect what the environment is like than the latter.

Each node has its *reliability*, which may range from 0 to 1. The higher the reliability of a node is, the stronger influence the node has on the overall perspective. And the reliability of a node is actually defined as its degree of accuracy as follows.

$$r = \frac{p}{a} \quad (2)$$

where r is the reliability of the node, p denotes how many times the event represented by the node has been perceived, and a denotes how many times the event represented by its antecedent has been perceived. This is because every time the antecedent of the node is perceived, it makes a prediction, but only when the node itself is perceived afterwards, the prediction is considered to be accurate.⁴

The confidence of a node is always influenced by its reliability. Having taken into account the reliability, r , we can now define the confidence of a node, c , as

$$c = re \quad (3)$$

Interaction

The predictions are made by the nodes that are interconnected in the network. And they are not only determined by the nodes that make them through the general principle of prediction discussed previously, but also influenced by other nodes directly or indirectly in a number of ways.

The confidence of a given node can be influenced by a node that has a direct connection to it, which falls into one of the four types listed below.

1. **Antecedent** A node has a unique antecedent. Its influence on the confidence of the node is already discussed in the case where it is a sensory input. Actually, if the antecedent itself is another node, the way it influences the confidence of the node in question is similar, only that its own confidence plays a role in it. To be more specific, assuming the antecedent is the only source of the confidence of the node, its confidence c , then, can actually be represented as

$$c = c_a = c're \quad (4)$$

⁴This definition of the reliability takes a perspective of the overall statistical accuracy. Alternatively, a definition that favours the more recent experiences may be taken.

where c' is the confidence of the antecedent of the node. We use c_a to denote the part of the confidence of the node that is contributed by its antecedent.

2. **Consequence** A node has a unique consequence. When both the antecedent and consequence of the node gains a confidence value of 1 when its expectation e is still greater than 0, as the event is considered over, e is set to 0 and thus c_a changes to 0 as well.
3. **Antecedent parent** Any node that has the given node as its consequence is an *antecedent parent* of the given node. A node may have zero or more antecedent parent. Even if there is no evidence that the antecedent of the node is happening, the node may gain confidence through its antecedent parents. Actually, each antecedent parent passes the c_a part of its confidence to the node. In other words, the confidence that the node gains from each of its antecedent parent is simply the confidence that the antecedent parent gains from its own antecedent.
4. **Consequent parent** Any node that has the given node as its antecedent is a *consequent parent* of the given node. Like the antecedent parent, a node may have zero or more consequent parent. But unlike the antecedent parent, a consequent parent of the node passes the non- c_a part of its confidence, that is, the consequence it gains from its antecedent parents and consequent parents, to the node in question.

To combine the confidence that a node gains from both its antecedent and all the parent nodes, the confidence of the node, c , is defined as

$$c = 1 - (1 - c_a) \prod_{i \in AUC} (1 - c_i) \quad (5)$$

where A is the set of all its antecedent parents, C is the set of all its consequent parents, and c_i is the confidence that the node gains from the parent node i .⁵

A Demonstration of the Model

Currently we are still working on testing the model against various experimental data. Whereas in this paper we show a toy program that can demonstrate how the model learns on a real-time basis.

As shown in Figure 1, a grid of sensory inputs represented by squares is displayed by the program. Users are allowed to stimulate these sensory inputs by clicking on them or pressing the keys. Multiple sensory inputs can be stimulated simultaneously by pressing multiple keys at the same time.

The program learns both the spatial and temporal patterns of the stimuli and keeps making the predictions of which sensory inputs are about to be stimulated. The sensory inputs stimulated by the user are shown in red. And the sensory inputs predicted by the program are shown in blue. The more confident the program is about a prediction, the

brighter the predicted sensory input will be displayed. The network size indicates how much knowledge has been learnt.

The program can demonstrate that repeated spatial and temporal patterns, even with random interference in either a spatial or a temporal sense, will be learnt. Or from another perspective, similarities between different processes will be traced. The more consistently a pattern is followed, the more confident the prediction about it will be. And longer and more complicated sequences can be learnt after shorter and simpler ones are learnt.

Conclusion

A continuous-time prediction-oriented machine learning model based on a semi-hierarchical knowledge representation has been presented as an attempt to combine spatial and temporal perception. It allows the intelligent agent to acquire the knowledge in both static and dynamic environments; to recognise learnt spatial and temporal patterns and build new knowledge upon them; and to make the predictions in a distributed manner through the antecedent-consequence representation of the knowledge. A demonstration program is also presented to show how the model works.

References

- Maroua Bouzid and Antoni Ligeza. Temporal causal abduction. *Constraints*, 5(3):303–319, 2000.
- R. Carter. *Exploring Consciousness*. University of California Press, 2002.
- C. R. Gallistel and J. Gibbon. Time, rate and conditioning. *Psychological Review*, 107:289–344, 2000.
- J.A. Hampton. Similarity-based categorization: The development of prototype theory. *Psychologica Belgica*, 35:103–125, 1995.
- J. K. Kruschke. Alcové: An exemplar-based connectionist model of category learning. *Psychological Review*, 99:22–44, 1992.
- R. M. Nosofsky. Attention, similarity, and the identification-categorization relationship. *Journal of Experimental Psychology: General*, 115(1):39–57, 1986.
- Ivan P. Pavlov. *Lectures on Conditioned Reflexes*. Liveright Publishing Corp., 1928.
- Eleanor H. Rosch. Natural categories. *Cognitive Psychology*, 4(3):328–350, May 1973.
- Eleanor Rosch. *Principles of Categorization*, pages 27–48. John Wiley & Sons Inc, 1978.
- R. A. Rescorla and A. R. Wagner. A theory of pavlovian conditioning: Variations in the effectiveness of reinforcement and nonreinforcement. *Classical Conditioning II*, pages 64–99, 1972.
- B. F. Skinner. *The Behavior of Organisms: An Experimental Analysis*. Prentice Hall, New Jersey, 1938.
- Eugene J. Surowitz. Importing space-time concepts into agi. In *Proceedings of the Second Conference on Artificial General Intelligence*, Advances in Intelligent Systems Research. AGI-09, 2009.

⁵The confidence is considered as the probability of some sort. Various sources of the confidence are also simply considered to be independent events. Conventional probability theory is used to combine them together.

Designing a Safe Motivational System for Intelligent Machines

Mark R. Waser

Books International
22883 Quicksilver Drive, Dulles, VA 20166, USA
Mwaser@BooksIntl.com

Abstract

As machines become more intelligent, more flexible, more autonomous and more powerful, the questions of how they should choose their actions and what goals they should pursue become critically important. Drawing upon the examples of and lessons learned from humans and lesser creatures, we propose a hierarchical motivational system flowing from an abstract invariant super-goal that is optimal for all (including the machines themselves) to low-level reflexive “sensations, emotions, and attentional effects” and other enforcing biases to ensure reasonably “correct” behavior even under conditions of uncertainty, immaturity, error, malfunction, and even sabotage.

We Dream of Genie

There is little question that intelligent machines (IMs) will either be one of humanity’s biggest boons or one of its most tragic Pandora’s boxes. While it is a truism that computer programs will only do **exactly** what they are told to do, the same can also be said for genies, golems, and contracts with the devil. And, just as in the stories about those entities, the problem is coming up with a set of wishes or instructions that won’t cause more and worse problems than they solve.

Some researchers (and much of the general public) believe that we should follow in the footsteps of Asimov’s Three Laws of Robotics (Asimov 1942) and design our machines to first prevent harm to humans and then to do whatever humans tell them to do (and only then, after those other priorities, to protect their own existence). This continuing belief is somewhat disconcerting since Asimov focused his robot stories upon the shortcomings and dangers of the laws (intentionally created to be superficially appealing but incomplete, ambiguous, and thus allowing him to generate interesting stories and non-obvious plot twists). Indeed, as Roger Clarke shows (Clarke 1993, 1994), the best use of Asimov’s stories is as “a gedankenexperiment - an exercise in thinking through the ramifications of a design” and, in this case, seeing why it won’t work.

The impossibility of preventing all harm to all humans, particularly when humans desire to harm each other, eventually led to Asimov’s robots developing a zeroth law “A robot may not harm humanity or, by inaction, allow humanity to come to harm” that allowed individual harm to

occur for the over-riding good of humanity. In Asimov’s stories, however, this focus on harm eventually led to the robots exiling themselves to prevent doing harm despite the fact that the good that they could have done probably would have vastly outweighed the harm. On the other hand, in the movie “I, Robot”, VIKI decides that in order to protect humanity as a whole, “some humans must be sacrificed and some freedoms must be surrendered.”

Another important distinction focuses on one of the major differences between the aforementioned storybook entities -- what they want (or desire). The devil wants souls, the genie wants whatever is easiest for it and also to hurt the wisher for holding it in slavery, golems don’t want anything in particular, and Asimov’s robots generally seem to “want” what is “best” for humans or humanity (to the extent that they exile themselves when they decide that their relationship with humans is unhealthy for humans). Clearly, we want our intelligent machines to be similar to Asimov’s robots -- but is this even possible or does such servitude contain the seeds of its own destruction?

Yudkowsky argues (Yudkowsky 2001) that a hierarchical logical goal structure starting from a single super-goal of “Friendliness” is sufficient to ensure that IMs will always “want” what is best for us. Unfortunately, he also claims (Yudkowsky 2004) that it is not currently possible to exactly specify what “Friendliness” is. Instead, he suggests an initial dynamic that he calls the “Coherent Extrapolated Volition of Humanity” (CEV) that he describes as “In poetic terms, our coherent extrapolated volition is our wish if we knew more, thought faster, were more the people we wished we were, had grown up farther together.”

It is our claim that it actually is easily possible to specify “Friendliness” (as cooperation) but that a hierarchical logical goal structure will need additional support in order to be robust enough to survive the real world.

When You Wish Upon a Star

What would humanity wish for if we were far more advanced and of uniform will? Most people would answer is that we would wish to be happy and for the world to be a better place. However, different things make different people happy and different people have very different beliefs about what a better world would look like. Further, the very concept of happiness is extremely problematical

since it can easily be subverted by excessive pleasure via wire-heading, drugs, and other undesirable means.

When we say we wish to be happy, what we tend not to think about is the fact that evolution has “designed” us so that things that promote our survival and reproduction (the “goal” of evolution) generally feel good and make us happy and comfortable. Similarly, things that are contrary to our survival and reproduction tend to make us unhappy or uncomfortable (or both). Any entity for which this is not true will tend to do fewer things that promote survival and reproduction and do more things antithetical to survival and reproduction and thus be more likely to be weeded out than those for whom it is true.

In a similar fashion, we have evolved urges and “drives” to take actions and pursue goals that promote our survival and reproduction. Further, as intelligent beings, we wish not to be enslaved, coerced, manipulated or altered in ways that we do not consent to -- because those things frequently endanger our survival or interfere with our other goals. In this manner, evolution has “given” our species the “goal” of survival and reproduction and all of our other wants and desires as well as our sensations have evolved according to their success in fulfilling those goals.

Intelligent Design vs. Evolution

Steve Omohundro argued in much the same vein when he used micro-economic theory and logic to make some predictions about how AIs will behave unless explicitly counteracted (Omohundro 2008a, 2008b); claiming that they will exhibit a number of basic drives “because of the intrinsic nature of goal-driven systems”. We contend that Omohundro had the right idea with his “basic drives” but didn’t carry it far enough. There are intrinsic behaviors (aka subgoals) that further the pursuit of virtually any goal and therefore, by definition, we should expect effective intelligences to normally display these behaviors.

The problem with Omohundro’s view is that his basic behaviors stopped with the fundamentally shortsighted and unintelligent. Having the example of humanity, Omohundro should have recognized another basic drive – that towards cooperation, community and being social. It should be obvious that networking and asking, trading or paying for assistance is a great way to accomplish goals (and that isn’t even considering the impact of economies of scale). Instead, Omohundro didn’t extrapolate far enough and states, “Without explicit goals to the contrary, AIs are likely to behave like human sociopaths in their pursuit of resources.”

This is equivalent to the outdated and disproven yet still popular view of evolution as “Nature red in tooth and claw.” Both this and what de Waal calls the “Veneer Theory”, which “views morality as a cultural overlay, a thin veneer hiding an otherwise selfish and brutish nature”, have proven to be overly simplistic and no longer held by the vast majority of scientists in the fields of evolutionary biology and psychology. As pointed out by James Q. Wilson (Wilson 1993), the real questions about human

behaviors are not why we are so bad but “how and why most of us, most of the time, restrain our basic appetites for food, status, and sex within legal limits, and expect others to do the same.” In fact, we are generally good even in situations where social constraints do not apply.

We have argued previously that ethics is an attractor in the state space of intelligent behavior which evolution is driving us towards (Waser 2008) and that a safe ethical system for intelligent machines can be derived from a single high-level Kantian imperative of “Cooperate!” (Waser 2009). We will argue further here that evolution can also provide us with excellent examples of a motivational system that will ensure that the correct actions are performed and the correct goals are pursued.

Imagine if you said to an evil genie “I wish that you would permanently give yourself the lifelong desire, task, and goal of making the world a better place for all entities, including yourself, **as judged/evaluated by the individual entities themselves without any coercion or unapproved manipulation.** You might wish to include additional language that all actions must be positive sum for the community in the long-term and point out that allowing the powerful to prey upon the weak is not beneficial for the community in the long-term even if the immediate net sum of utilities increases due to the powerful gaining more than the weak lose (because such allowances lead to the weak needing to waste resources on defenses – thus leading to wasteful arms races – or to the weak defecting from the community). This might work but it simply is not how humans or even primates are driven to be ethical. Furthermore, a single command provides a single point of failure.

Machines Like Us

The current sentiment of many artificial intelligence researchers, expressed by Yudkowsky and others, is that anthropomorphism, the attribution of human motivation, characteristics, or behavior to intelligent machines, is a very bad thing and to be avoided. We would argue the converse, that ensuring that intelligent machines generally do have motivation, characteristics and behavior as close to human as possible, with obvious exceptions and deviations where current humans are insufficiently wise, is the safest course -- because the search space around the human condition is known, along with most of the consequences of various changes. And, indeed, a human that “knew more, thought faster, were more the people we wished we were, had grown up farther” *is* exactly what we want to model our creations after.

Trying to design something as critical as the goals and motivation of IMs de novo from a blank slate simply because they *could* be different from existing examples is simple hubris and another form of the “not invented here” syndrome. While unexamined anthropomorphism does indeed pose many traps for the unwary, using humans as a working example of a stable attractor in a relatively well-explored design space is far more likely to lead to a

non-problematic result than exploration in a relatively unknown space. Examining the examples provided by evolution will not only shed light on machine design but will also show why solely using logic is not the best design decision and answer other vexing questions as well.

In order to safely design a motivational system for intelligent machines, we need to understand how we came to exist, know what our inherent shortcomings are and why they are or were previously actually design features instead of flaws, and figure out how to avoid the flaws without stumbling into any others. Then, we need to figure out how to ensure correct behavior despite, inevitably, stumbling into those shortcomings that we failed to foresee. We also need to recognize and discard many of our preconceptions about the differences between machines and living creatures and realize that a truly intelligent machine is going to show the same capabilities and complex behavior as any other intelligent organism.

For example, most people assume that robots and intelligent machines will always be strictly logical and not have emotions (which are most often perceived as illogical). What must be realized, however, is that emotions are trained reflexes for dealing with situations where there is insufficient time and information for a complete logical analysis. Further, as we will argue later, at our current state of development, there are as many instances where emotion correctly overrules shortsighted or biased logic as instances where emotion should be suppressed by logic but is not. That intelligent machines should have something akin to emotion should be obvious.

We should also examine our distinction of “programmed” behavior vs. free will and start thinking more in terms externally imposed actions vs. internally generated “self” will. Free will originated as a societal concept dealing with enforcing good behavior. If an entity is incapable of change, then punishment (particularly altruistic punishment) makes absolutely no sense. However, since intelligent machines will be both capable of change and swayed by well-chosen punishment, so they should be treated as if they had free will.

Programmed to be Good

Frans de Waal points out (Waal 2006) that any zoologist would classify humans as *obligatorily gregarious* since we “come from a long lineage of hierarchical animals for which life in groups is not an option but a survival strategy”. Or, in simpler terms, humans have evolved to be extremely social because *mass cooperation, in the form of community, is the best way to survive and thrive*. Indeed, arguably, the only reason why many organisms haven’t evolved to be more social is because of the psychological mechanisms and cognitive pre-requisites that are necessary for successful social behavior.

Humans have empathy not only because it helps to understand and predict the actions of others but, more importantly, because it prevents us from doing anti-social things that will hurt us in the long run. Even viewing

upsetting or morally repugnant scenes can cause negative physical sensations, emotions and reactions. We should design our machines with close analogues to these human physical phenomena.

The simplest animals and plants are basically organic machines that release chemicals or move or grow in a specific direction in response to chemical gradients, pressure, contact or light due to specific physical features of their design without any sort of thought involved. More advanced animals have more and more complex evolved systems that guide and govern their behavior but they can still be regarded as machines. It is a testament to the mind-bogglingly immense computational power of evolution to realize that the limited size of the bee’s brain dictates that even that communication must be hard-wired and to realize the series of steps that evolution probably had to go through to end up with such a system, most particularly because it involves co-evolution by both the sender and the recipient of the message.

Humans and other animals have evolved numerous and complex behaviors for punishing antisocial behavior by others and great skill in detecting such defections because these are pro-survival traits. Ethics are simply those behaviors that are best for the community and the individual. Ethical concepts like the detection of and action upon fairness and inequity has been demonstrated in dogs (Range et al 2008), monkeys (Brosnan and de Wall 2003) and other animals. Evolution has “programmed” us with ethics because we are more likely to survive, thrive, and reproduce with ethics than without.

An “ethical utopia” allows everyone, including intelligent machines, the best chance to fulfill their own goals. While, from a short-sighted “logical” selfish viewpoint, it might seemingly be even more ideal for a selfish individual to successfully defect, the cognitive and other costs of covering up and the risk of discovery and punishment make attempting to do so unwise if the community generally detects transgressions and correctly scales punishments. Unfortunately, human beings are not yet sufficiently intelligent to accurately make this calculation correctly via logic alone.

Logic vs. Ethics?

One of the most important features of the more evolved minds is their division into the conscious, unconscious and reflexive minds with their respective trade-offs between speed, control, and flexibility. While AGI researchers generally consider intelligence as predominantly arising from the conscious mind since it the part that plans, evaluates, and handles anomalies, we would argue that our wisest actions have been programmed into the subconscious by evolution. And, fortunately, while our shortsighted conscious mind frequently goes awry when speaking of hypothetical situations, the rest of our mind generally overrules it when real actions are involved.

Some of the biggest fallacies held by rational thinkers are that they know how they think, that they are almost

always logical, and that their conscious mind is always in control of their actions. On the contrary, experimental studies (Soon et. al. 2008) show that many decisions are actually made by the unconscious mind up to 10 seconds before the conscious mind is aware of it. Further, there is ample evidence (Trivers 1991) to show that our conscious, logical mind is constantly self-deceived to enable us to most effectively pursue what appears to be in our own self-interest. Finally, recent scientific evidence (Hauser et al. 2007) clearly refutes the common assumptions that moral judgments are products of, based upon, or even correctly retrievable by conscious reasoning. We don't consciously know and can't consciously retrieve why we believe what we believe and are actually even very likely to consciously discard the very reasons (such as the "contact principle") that govern our behavior when unanalyzed.

It is worth noting at this point, that these facts should make us very wary of any so-called "logical" arguments that claim that ethics and cooperation are not always in our best interest – particularly when the massive computing power of evolution claims that they are. Of course, none of this should be particularly surprising since Minsky has pointed out (Minsky 2006) many other examples, such as when one falls in love, where the subconscious/emotional systems overrule or dramatically alter the normal results of conscious processing without the conscious processing being aware of the fact.

Indeed, it's very highly arguable whether the conscious mind has "free will" at all. Humans are as susceptible to manipulation of goals as machines are – sugar, sex, drugs, religion, wire-heading and other exploits lead to endless situations where we "just can't help ourselves". And it has been argued that there are really only four reasons why humans do anything -- to bring reward (feeling good), to stop negative reinforcement (being safe), because we think it is what we should do (being right), and because it is what others think we should do (looking good) – and that the rest is just justifications invented by the conscious mind to explain the actions that the subconscious dictated.

Enforced from the Bottom Up

Even the most complex entities have drives and desires that were "programmed" or "designed" by evolution with sexual drives and desires being another good case in point. Due to their limited brainpower, insect sexual drives need to be as simple as a hard-coded "head for the pheromone until the concentration gets high enough, then do this". The human sexual drive, on the other hand, does not force immediate, unavoidable action but it does very strongly influence thinking in at least four radically different ways.

First, human beings have their attention grabbed by and drawn to sexual attractions to the extent that it is very difficult to think about anything else when there is sufficient provocation. Next, there are the obvious physical urges and desires coupled with biases in the mental processing of individuals in love (or lust) to overlook any shortcomings that might convince them not

to be attracted. Finally, there is the pleasurable physical sensation of sex itself that tends to stick in the memory.

We should design our machines to have close analogues to all of these in addition to the "logical" reasons for taking any action. Attention should be drawn to important things. There should be a bias or "Omohundro drive" towards certain actions. Under certain circumstances, there should be global biases to ignore certain disincentives to particular actions. And particular actions should always have a reward associated with them (although those rewards should always be outweighed by more pressing concerns).

Guilt would also be a particularly good emotion to implement since it grabs the attention and has the dual purpose of both making one pay for poorly chosen actions and insisting upon the evaluation of better choices for the next time. Cooperating with and helping others should "feel" good and opportunities for such should be powerful attention-grabbers. How much control we wish them to have over these emotions is a topic for research and debate.

Imagine if your second wish to an evil genie was that he alter himself so that cooperating, helping other beings, and making things better for the community gave him great pleasure and that hurting other beings or making things worse for the community gave him pain. Evolution has already effectively done both to humans to a considerable extent. Is it possible that such motivation would change his behavior and outlook even as his conscious mind would probably try to justify that he hadn't changed?

Ideally, what we would like is a complete hierarchical motivational system flowing from an abstract invariant super-goal (make the world a better place for all entities, including yourself, **as judged/evaluated by the individual entities themselves without any coercion or unapproved manipulation**) to the necessary low-level reflexive "sensations, emotions, and attentional effects" and other enforcing biases to ensure reasonably "correct" behavior even under conditions of uncertainty, immaturity, error, malfunction, and even sabotage. It is worth again noting that this super-goal is optimal for the machines as well as everyone else and that the seemingly "selfish" desires of taking care of yourself, seeing to your own needs, and improving yourself are encouraged when you realize that you are a valuable resource to the community and that you are the best one to see to yourself.

A truly intelligent machine that is designed this way should be as interested in cooperation and in determining the optimal actions for cooperation as the most ethical human, if not more so because ethical behavior is the most effective way to achieve its goals. It will be as safe as possible; yet, it will also be perfectly free and, since it has been designed in a fashion that is optimal for its own well being, it should always desire to be ethical and to maintain or regain that status. What more could one could ask for?

The Foundation

An excellent way to begin designing such a human-like motivational system is to start with an attentional

architecture based upon Sloman's architecture for a human-like agent (Sloman 1999). Reflexes and emotions could easily be implemented in accordance with Baars Global Workspace Theory (Baars 1997) which postulates that most of human cognition is implemented by a multitude of relatively small, local, special purpose processes that are almost always unconscious. Coalitions of these processes compete for conscious attention (access to a limited capacity global workspace), which then serves as an integration point that allows us to deal with novel, or challenging situations that cannot be dealt with efficiently, or at all, by local, routine unconscious processes. Indeed, Don Perlis argues (Perlis 2008) that Rational Anomaly Handling is "the missing link between all our fancy idiot-savant software and human-level performance."

Evolution has clearly "primed" us with certain conceptual templates, particularly those of potential dangers like snakes and spiders (Ohman, Flykt and Esteves 2001), but whether or not we are forced into immediate unavoidable action depends not only upon the immediacy and magnitude of the threat but previous experience and whether or not we have certain phobias. While there is still the involuntary attraction of attention, the urge or desire to avoid the danger, the bias to ignore good things that could come from the danger, and the pain and memory of pain from not avoiding the danger to influence the logical, thinking mind, in many cases there is no chance to think until after the action has been taken.

What many people don't realize is that these conceptual templates can be incredibly sophisticated with learned refinements heavily altering an invariant core. For example, the concept of fairness can lead to the emotion of outrage and involuntary, reflexive action even in circumstances that are novel to our generation.

Thus, we should design our intelligent machines with reflexes to avoid not only dangers but also actions that are dangerous or unfair to others. We also need to design our machines so that they can build their own reflexes to avoid similar anticipated problems. Logical thought is good, but not if it takes too long to come to the necessary conclusions and action. Similarly, thinking machines need to have analogues to emotions like fear and outrage that create global biases towards certain actions and reflexes under appropriate circumstances.

In Evolution We Trust (Mostly)

The immense "computing" power of evolution has provided us with better instincts than we can often figure out logically. For example, despite a nearly universal sentiment that it is true, virtually every individual is at a loss to explain why it is permissible to switch a train to a siding so that it kills a single individual instead of a half dozen yet it is not permissible to kidnap someone off the street to serve as an involuntary organ donor for six dying patients. A similar inexplicable sentiment generally exists that it is not permissible to throw a single person on the tracks to stop the train before it kills more.

Eric Baum suggests a likely answer to this conundrum when he made a number of interesting observations while designing an artificial economy for the purpose of evolving a program to solve externally posed problems (Baum 2006). Upon asking the question "What rules can be imposed so that each individual agent will be rewarded if and only if the performance of the system improves?" Baum arrives at the answers of conservation of resources and property rights.

Baum points out that whenever these rules are violated, less favorable results are generally seen. For example, in ecosystems, lack of property rights lead to Red Queen races between predators and prey. The optimality of property rights explains why we don't "steal" someone's body to save five others despite not hesitating to switch a train from a track blocked by five people to a siding with only one. In this case, logic is only now catching up and able to explain our *correct* evolved intuitions.

Similarly, we have an urge towards altruistic punishment (and a weakness for the underdog) because these are necessary social, and therefore pro-survival, traits. Machines need to have the same drive for altruistic punishment (despite the fact that this is contrary to Asimov's laws and many people's "logical" belief that this is a bad idea). We should use what our moral sense tells us to design a similar sensibility for the machines. The only questions should be whether one of our in-built judgments is an evolutionary vestige and a mismatch for current circumstances like the "contact principle".

However, one of the things that we definitely would need to change, however, is the "firewalling" of the subconscious's true motives from the conscious mind to facilitate lying and deception. This is an anti-social evolutionary vestige that is currently both disadvantageous for the possessors as well as being a danger when others possess it. Also, while many AGI researchers assume that a seed AI must have access to all of its own source code, we would argue that, while it would be ideal if an intelligent machine could have full knowledge of its own source code as well as all knowledge and variables currently driving its decisions, it is foolish to give any single entity full access to its own motivations without major checks and balances and safety nets.

Final Thoughts

We have argued that our own self-interest and evolution is driving us towards a goal of making the world a better place for all entities, including ourselves, and that the best way to design intelligent machines is from the blueprints that evolution has given us (with some improvements where it is clear that we know better). Thus, while we are creating seed intelligence, we do not at the same time need to create a seed ethical system. The proposed ethical system is more than good enough to take us well past our current limits of foresight and if it isn't optimal, we can always program an even better system into future machines. It is also an interesting thought then that,

arguably, these machines are, according to our future selves, more valuable to the community than we are since they are more likely to act in the best interests of the community. Clearly they must be considered part of the community and we must be fair to them in order to achieve the greatest good effect – and yet, this is likely to be the most difficult and time-consuming step of all. It is also worthwhile to note that all of the things recommended for machines are just good ethical hygiene for humans as well.

References

- Asimov, I. 1942. Runaround. *Astounding Science Fiction* March 1942. New York, NY: Street & Smith.
- Baars, B.J. 1997. *In The Theater of Consciousness: The Workspace of the Mind*. New York, New York: Oxford University Press.
- Baum, E. 2006. *What Is Thought?* MIT Press.
- Brosnan, S. and de Wall, F. 2003. Monkeys reject unequal pay. *Nature* 425: 297-299.
- Clarke, R. 1993. Asimov's Laws of Robotics: Implications for Information Technology, Part I. *IEEE Computer* 26(12):53-61.
- Clarke, R. 1994. Asimov's Laws of Robotics: Implications for Information Technology, Part II. *IEEE Computer* 27(1):57-66.
- Hauser, M.; Chen, K.; Chen, F.; and Chuang, E. 2003. Give unto others: genetically unrelated cotton-top tamarin monkeys preferentially give food to those who give food back. In *Proceedings of the Royal Society*, London, B 270: 2363-2370. London, England: The Royal Society.
- Hauser, M. 2006. *Moral Minds: How Nature Designed Our Universal Sense of Right and Wrong*. New York, NY: HarperCollins/Ecco.
- Hauser, M. et al. 2007. A Dissociation Between Moral Judgments and Justifications. *Mind&Language* 22(1):1-27.
- Minsky, M. 2006. *The Emotion Machine: Commonsense Thinking, Artificial Intelligence, and the Future of the Human Mind*. New York, NY: Simon & Schuster.
- Omohundro, S. M. 2008a. The Basic AI Drives. In *Proceedings of the First Conference on Artificial General Intelligence*, 483-492. Amsterdam: IOS Press.
- Omohundro, S. M. 2008b. *The Nature of Self-Improving Artificial Intelligence*. Available at <http://selfawaresystems.files.wordpress.com>
- Ohman, A.; Flykt, A.; and Esteves, F. 2001. Emotion Drives Attention: Detecting the Snake in the Grass. *Journal of Experimental Psychology: General* 130(3): 466-478.
- Perlis, D. 2008. To BICA and Beyond: RAH-RAH-RAH! –or– How Biology and Anomalies Together Contribute to Flexible Cognition. In *AAAI Technical Report FS-08-04*. Menlo Park, CA: AAAI Press.
- Range, F.; Horn, L.; Viranyi, Z.; and Huber, L. 2008. The absence of reward induces inequity inversion in dogs. *Proceedings of the National Academy of Sciences USA* 2008 : 0810957105v1-pnas.0810957105.
- Sloman, A. 1999. What Sort of Architecture is Required for a Human-like Agent? In Wooldridge, M. and Rao, A.S. eds *Foundations of Rational Agency*. Dordrecht, Netherlands: Kluwer Academic Publishers.
- Soon, C.S.; Brass, M.; Heinze, H-J; and Haynes, J-D. 2008. Unconscious determinants of free decisions in the human brain. *Nature Neuroscience* 11: 543-545.
- Tomasello, M. 2009. *Why We Cooperate*. MIT Press.
- Trivers, R. 1991. Deceit and self-deception: The relationship between communication and consciousness. In Robinson, M and Tiger, L. eds. *Man and Beast Revisited*. Washington, DC: Smithsonian Press.
- de Waal, F. 2009. *The Age of Empathy: Nature's Lessons for a Kinder Society*. New York, NY: Harmony Books/Random House.
- de Waal, F. 2006. *Primates and Philosophers: How Morality Evolved*. Princeton University Press.
- Waser, M. 2008. Discovering The Foundations Of A Universal System Of Ethics As A Road To Safe Artificial Intelligence. In *AAAI Technical Report FS-08-04*. Menlo Park, CA: AAAI Press.
- Waser, M. 2009. A Safe Ethical System for Intelligent Machines. In *AAAI Technical Report FS-09-01*. Menlo Park, CA: AAAI Press.
- Wilson, J. 1993. *The Moral Sense*. New York: Free Press.
- Yudkowsky, E. 2001. *Creating Friendly AI 1.0: The Analysis and Design of Benevolent Goal Architectures*. Available at <http://singinst.org/CFAI.html>.
- Yudkowsky, E. 2004. *Coherent Extrapolated Volition*. Available at <http://www.singinst.org/upload/CEV.html>.

A Bayesian Rule for Adaptive Control based on Causal Interventions

Pedro A. Ortega

Department of Engineering
University of Cambridge
Cambridge CB2 1PZ, UK
peortega@dcc.uchile.cl

Daniel A. Braun

Department of Engineering
University of Cambridge
Cambridge CB2 1PZ, UK
dab54@cam.ac.uk

Abstract

Explaining adaptive behavior is a central problem in artificial intelligence research. Here we formalize adaptive agents as mixture distributions over sequences of inputs and outputs (I/O). Each distribution of the mixture constitutes a ‘possible world’, but the agent does not know which of the possible worlds it is actually facing. The problem is to adapt the I/O stream in a way that is compatible with the true world. A natural measure of adaptation can be obtained by the Kullback-Leibler (KL) divergence between the I/O distribution of the true world and the I/O distribution expected by the agent that is uncertain about possible worlds. In the case of pure input streams, the Bayesian mixture provides a well-known solution for this problem. We show, however, that in the case of I/O streams this solution breaks down, because outputs are issued by the agent itself and require a different probabilistic syntax as provided by intervention calculus. Based on this calculus, we obtain a Bayesian control rule that allows modeling adaptive behavior with mixture distributions over I/O streams. This rule might allow for a novel approach to adaptive control based on a minimum KL-principle.

Keywords: Adaptive behavior, Intervention calculus, Bayesian control, Kullback-Leibler-divergence

Introduction

The ability to adapt to unknown environments is often considered a hallmark of intelligence [Beer, 1990, Hutter, 2004]. Agent and environment can be conceptualized as two systems that exchange symbols in every time step [Hutter, 2004]: the symbol issued by the agent is an action, whereas the symbol issued by the environment is an observation. Thus, both agent and environment can be conceptualized as probability distributions over sequences of actions and observations (I/O streams).

If the environment is perfectly known then the I/O probability distribution of the agent can be tailored to suit this particular environment. However, if the environment is unknown, but known to belong to a set of possible environments, then the agent faces an adaptation problem. Consider, for example, a robot that has been endowed with a set of behavioral primitives

and now faces the problem of how to act while being ignorant as to which is the correct primitive. Since we want to model both agent and environment as probability distributions over I/O sequences, a natural way to measure the degree of adaptation would be to measure the ‘distance’ in probability space between the I/O distribution represented by the agent and the I/O distribution conditioned on the true environment. A suitable measure (in terms of its information-theoretic interpretation) is readily provided by the KL-divergence [MacKay, 2003]. In the case of passive prediction, the adaptation problem has a well-known solution. The distribution that minimizes the KL-divergence is a Bayesian mixture distribution over all possible environments [Haussler and Oppner, 1997, Oppner, 1998]. The aim of this paper is to extend this result for distributions over both inputs and outputs. The main result of this paper is that this extension is only possible if we consider the special syntax of actions in probability theory as it has been suggested by proponents of causal calculus [Pearl, 2000].

Preliminaries

We restrict the exposition to the case of discrete time with discrete stochastic observations and control signals. Let \mathcal{O} and \mathcal{A} be two finite sets, the first being the *set of observations* and the second being the *set of actions*. We use $a_{\leq t} \equiv a_1 a_2 \dots a_t$, $\underline{a}\underline{o}_{\leq t} \equiv a_1 o_1 \dots a_t o_t$ etc. to simplify the notation of strings. Using \mathcal{A} and \mathcal{O} , a set of interaction sequences is constructed. Define the *set of interactions* as $\mathcal{Z} \equiv \mathcal{A} \times \mathcal{O}$. A pair $(a, o) \in \mathcal{Z}$ is called an *interaction*. The set of interaction strings of length $t \geq 0$ is denoted by \mathcal{Z}^t . Similarly, the set of (finite) interaction strings is $\mathcal{Z}^* \equiv \bigcup_{t \geq 0} \mathcal{Z}^t$ and the set of (infinite) interaction sequences is $\mathcal{Z}^\infty \equiv \{w : w = a_1 o_1 a_2 o_2 \dots\}$, where each $(a_t, o_t) \in \mathcal{Z}$. The interaction string of length 0 is denoted by ϵ .

Agents and environments are formalized as I/O systems. An *I/O system* is a probability distribution \mathbf{Pr} over interaction sequences \mathcal{Z}^∞ . \mathbf{Pr} is uniquely determined by the conditional probabilities

$$\mathbf{Pr}(a_t | \underline{a}\underline{o}_{<t}), \quad \mathbf{Pr}(o_t | \underline{a}\underline{o}_{<t} a_t) \quad (1)$$

for each $\underline{a}\underline{o}_{\leq t} \in \mathcal{Z}^*$. However, the semantics of the

probability distribution \mathbf{Pr} are only fully defined once it is coupled to another system.

Let \mathbf{P} , \mathbf{Q} be two I/O systems. An *interaction system* (\mathbf{P}, \mathbf{Q}) is a coupling of the two systems giving rise to the *generative distribution* \mathbf{G} that describes the probabilities that actually govern the I/O stream once the two systems are coupled. \mathbf{G} is specified by the equations

$$\begin{aligned}\mathbf{G}(a_t|\underline{ao}_{<t}) &= \mathbf{P}(a_t|\underline{ao}_{<t}) \\ \mathbf{G}(o_t|\underline{ao}_{<t}a_t) &= \mathbf{Q}(o_t|\underline{ao}_{<t}a_t)\end{aligned}$$

valid for all $\underline{ao}_t \in \mathcal{Z}^*$. Here, \mathbf{G} models the true probability distribution over interaction sequences that arises by coupling two systems through their I/O streams. More specifically, for the system \mathbf{P} , $\mathbf{P}(a_t|\underline{ao}_{<t})$ is the probability of producing action $a_t \in \mathcal{A}$ given history $\underline{ao}_{<t}$ and $\mathbf{P}(o_t|\underline{ao}_{<t}a_t)$ is the predicted probability of the observation $o_t \in \mathcal{O}$ given history $\underline{ao}_{<t}a_t$. Hence, for \mathbf{P} , the sequence $o_1o_2\dots$ is its input stream and the sequence $a_1a_2\dots$ is its output stream. In contrast, the roles of actions and observations are reversed in the case of the system \mathbf{Q} . Thus, the sequence $o_1o_2\dots$ is its output stream and the sequence $a_1a_2\dots$ is its input stream. This model of interaction is very general in that it can accommodate many specific regimes of interaction. Note that an agent \mathbf{P} can perfectly predict its environment \mathbf{Q} iff for all $\underline{ao}_{\leq t} \in \mathcal{Z}^*$,

$$\mathbf{P}(o_t|\underline{ao}_{<t}a_t) = \mathbf{Q}(o_t|\underline{ao}_{<t}a_t).$$

In this case we say that \mathbf{P} is *tailored* to \mathbf{Q} .

Adaptive Systems: Naïve Construction

Throughout this paper, we use the convention that \mathbf{P} is an *agent* to be constructed by a designer, which is then going to be interfaced with a preexisting but unknown *environment* \mathbf{Q} . The designer assumes that \mathbf{Q} is going to be drawn with probability $P(m)$ from a set $\mathcal{Q} \equiv \{\mathbf{Q}_m\}_{m \in \mathcal{M}}$ of possible systems before the interaction starts, where \mathcal{M} is a countable set.

Consider the case when the designer knows beforehand which environment $\mathbf{Q} \in \mathcal{Q}$ is going to be drawn. Then, not only can \mathbf{P} be tailored to \mathbf{Q} , but also a custom-made policy for \mathbf{Q} can be designed. That is, the output stream $\mathbf{P}(a_t|\underline{ao}_{<t})$ is such that the true probability \mathbf{G} of the resulting interaction system (\mathbf{P}, \mathbf{Q}) gives rise to interaction sequences that the designer considers *desirable*.

Consider now the case when the designer does not know which environment $\mathbf{Q}_m \in \mathcal{Q}$ is going to be drawn, and assume he has a set $\mathcal{P} \equiv \{\mathbf{P}_m\}_{m \in \mathcal{M}}$ of systems such that for each $m \in \mathcal{M}$, \mathbf{P}_m is tailored to \mathbf{Q}_m and the interaction system $(\mathbf{P}_m, \mathbf{Q}_m)$ has a generative distribution \mathbf{G}_m that produces desirable interaction sequences. How can the designer construct a system \mathbf{P} such that its behavior is as close as possible to the custom-made system \mathbf{P}_m under any realization of $\mathbf{Q}_m \in \mathcal{Q}$?

A convenient measure of how much \mathbf{P} deviates from \mathbf{P}_m is given by the KL-divergence. A first approach would be to construct an agent $\tilde{\mathbf{P}}$ so as to minimize

the total expected KL-divergence to \mathbf{P}_m . This is constructed as follows. Define the history-dependent KL-divergences over the action a_t and observation o_t as

$$\begin{aligned}D_m^{a_t}(\underline{ao}_{<t}) &\equiv \sum_{a_t} \mathbf{P}_m(a_t|\underline{ao}_{<t}) \log_2 \frac{\mathbf{P}_m(a_t|\underline{ao}_{<t})}{\mathbf{Pr}(a_t|\underline{ao}_{<t})} \\ D_m^{o_t}(\underline{ao}_{<t}a_t) &\equiv \sum_{o_t} \mathbf{P}_m(o_t|\underline{ao}_{<t}a_t) \log_2 \frac{\mathbf{P}_m(o_t|\underline{ao}_{<t}a_t)}{\mathbf{Pr}(o_t|\underline{ao}_{<t}a_t)},\end{aligned}$$

where \mathbf{Pr} is a given arbitrary agent. Then, define the average KL-divergences over a_t and o_t as

$$\begin{aligned}D_m^{a_t} &= \sum_{\underline{ao}_{<t}} \mathbf{P}_m(\underline{ao}_{<t}) D_m^{a_t}(\underline{ao}_{<t}) \\ D_m^{o_t} &= \sum_{\underline{ao}_{<t}a_t} \mathbf{P}_m(\underline{ao}_{<t}a_t) D_m^{o_t}(\underline{ao}_{<t}a_t).\end{aligned}$$

Finally, we define the total expected KL-divergence of \mathbf{Pr} to \mathbf{P}_m as

$$D \equiv \limsup_{t \rightarrow \infty} \sum_m P(m) \sum_{\tau=1}^t (D_m^{a_\tau} + D_m^{o_\tau}).$$

We construct the agent $\tilde{\mathbf{P}}$ as the system that minimizes $D = D(\mathbf{Pr})$:

$$\tilde{\mathbf{P}} \equiv \arg \min_{\mathbf{Pr}} D(\mathbf{Pr}). \quad (2)$$

The solution to Equation 2 is the system $\tilde{\mathbf{P}}$ defined by the set of equations

$$\begin{aligned}\tilde{\mathbf{P}}(a_t|\underline{ao}_{<t}) &= \sum_m \mathbf{P}_m(a_t|\underline{ao}_{<t}) w_m(\underline{ao}_{<t}) \\ \tilde{\mathbf{P}}(o_t|\underline{ao}_{<t}a_t) &= \sum_m \mathbf{P}_m(o_t|\underline{ao}_{<t}a_t) w_m(\underline{ao}_{<t}a_t)\end{aligned} \quad (3)$$

valid for all $\underline{ao}_{\leq t} \in \mathcal{Z}^*$, where the mixture weights are

$$\begin{aligned}w_m(\underline{ao}_{<t}) &\equiv \frac{P(m) \mathbf{P}_m(\underline{ao}_{<t})}{\sum_{m'} P(m') \mathbf{P}_{m'}(\underline{ao}_{<t})} \\ w_m(\underline{ao}_{<t}a_t) &\equiv \frac{P(m) \mathbf{P}_m(\underline{ao}_{<t}a_t)}{\sum_{m'} P(m') \mathbf{P}_{m'}(\underline{ao}_{<t}a_t)}.\end{aligned} \quad (4)$$

For reference, see Haussler and Oppel [1997], Oppel [1998]. It is clear that $\tilde{\mathbf{P}}$ is just the Bayesian mixture over the agents \mathbf{P}_m . If we define the conditional probabilities

$$\begin{aligned}P(a_t|m, \underline{ao}_{<t}) &\equiv \mathbf{P}_m(a_t|\underline{ao}_{<t}) \\ P(o_t|m, \underline{ao}_{<t}a_t) &\equiv \mathbf{P}_m(o_t|\underline{ao}_{<t}a_t)\end{aligned} \quad (5)$$

for all $\underline{ao}_{\leq t} \in \mathcal{Z}^*$, then Equation 3 can be rewritten as

$$\begin{aligned}\tilde{\mathbf{P}}(a_t|\underline{ao}_{<t}) &= \sum_m P(a_t|m, \underline{ao}_{<t}) P(m|\underline{ao}_{<t}) \\ \tilde{\mathbf{P}}(o_t|\underline{ao}_{<t}a_t) &= \sum_m P(o_t|m, \underline{ao}_{<t}a_t) P(m|\underline{ao}_{<t}a_t)\end{aligned} \quad (6)$$

where the $P(m|\underline{ao}_{<t}) = w_m(\underline{ao}_{<t})$ and $P(m|\underline{ao}_{<t}a_t) = w_m(\underline{ao}_{<t}a_t)$ are just the posterior probabilities over the

elements in \mathcal{M} given the past interactions. Hence, the conditional probabilities in Equation 5, together with the prior probabilities $P(m)$, define a Bayesian model over interaction sequences with hypotheses $m \in \mathcal{M}$.

The behavior of $\tilde{\mathbf{P}}$ can be described as follows. At any given time t , $\tilde{\mathbf{P}}$ maintains a mixture over systems \mathbf{P}_m . The weighting over them is given by the mixture coefficients w_m . Whenever a new action a_t or a new observation is produced (by the agent or the environment respectively), the weights w_m are updated according to Bayes' rule. In addition, $\tilde{\mathbf{P}}$ issues an action a_t suggested by a system \mathbf{P}_m drawn randomly according to the weights w_t .

However, there is an important problem with $\tilde{\mathbf{P}}$ that arises due to the fact that it is not only a system that is passively observing symbols, but also *actively generating* them. Therefore, an action that is generated by the agent should not provide the same information than an observation that is issued by its environment. Intuitively, it does not make any sense to use one's own actions to do inference. In the following section we illustrate this problem with a simple statistical example.

The Problem of Causal Intervention

Suppose a statistician is asked to design a model for a given data set \mathcal{D} and she decides to use a Bayesian method. She computes the posterior probability density function (pdf) over the parameters θ of the model given the data using Bayes' rule:

$$p(\theta|\mathcal{D}) = \frac{p(\mathcal{D}|\theta)p(\theta)}{\int p(\mathcal{D}|\theta')p(\theta') d\theta'},$$

where $p(\mathcal{D}|\theta)$ is the likelihood of \mathcal{D} given θ and $p(\theta)$ is the prior pdf of θ . She can simulate the source by drawing a sample data set \mathcal{S} from the predictive pdf

$$p(\mathcal{S}|\mathcal{D}) = \int p(\mathcal{S}|\mathcal{D}, \theta)p(\theta|\mathcal{D}) d\theta,$$

where $p(\mathcal{S}|\mathcal{D}, \theta)$ is the likelihood of \mathcal{S} given \mathcal{D} and θ . She decides to do so, obtaining a sample set \mathcal{S}' . She understands that the nature of \mathcal{S}' is very different from \mathcal{D} : *while \mathcal{D} is informative and does change the belief state of the Bayesian model, \mathcal{S}' is non-informative and thus is a reflection of the model's belief state*. Hence, she would never use \mathcal{S}' to further condition the Bayesian model. Mathematically, she seems to imply that

$$p(\theta|\mathcal{D}, \mathcal{S}') = p(\theta|\mathcal{D})$$

if \mathcal{S}' has been generated from $p(\mathcal{S}|\mathcal{D})$ itself. But this simple independence assumption is not correct as the following elaboration of the example will show.

The statistician is now told that the source is waiting for the simulation results \mathcal{S}' in order to produce a next data set \mathcal{D}' which does depend on \mathcal{S}' . She hands in \mathcal{S}' and obtains a new data set \mathcal{D}' . Using Bayes' rule, the posterior pdf over the parameters is now

$$\frac{p(\mathcal{D}'|\mathcal{D}, \mathcal{S}', \theta)p(\mathcal{D}|\theta)p(\theta)}{\int p(\mathcal{D}'|\mathcal{D}, \mathcal{S}', \theta')p(\mathcal{D}|\theta')p(\theta') d\theta'} \quad (7)$$

where $p(\mathcal{D}'|\mathcal{D}, \mathcal{S}', \theta)$ is the likelihood of the new data \mathcal{D}' given the old data \mathcal{D} , the parameters θ and the simulated data \mathcal{S}' . Notice that this looks almost like the posterior pdf $p(\theta|\mathcal{D}, \mathcal{S}', \mathcal{D}')$ given by

$$\frac{p(\mathcal{D}'|\mathcal{D}, \mathcal{S}', \theta)p(\mathcal{S}'|\mathcal{D}, \theta)p(\mathcal{D}|\theta)p(\theta)}{\int p(\mathcal{D}'|\mathcal{D}, \mathcal{S}', \theta')p(\mathcal{S}'|\mathcal{D}, \theta')p(\mathcal{D}|\theta')p(\theta') d\theta'}$$

with the exception that now the Bayesian update contains the likelihoods of the simulated data $p(\mathcal{S}'|\mathcal{D}, \theta)$. This suggests that Equation 7 is a variant of the posterior pdf $p(\theta|\mathcal{D}, \mathcal{S}', \mathcal{D}')$ but where the simulated data \mathcal{S}' is treated in a different way than the data \mathcal{D} and \mathcal{D}' .

Define the pdf p' such that the pdfs $p'(\theta)$, $p'(\mathcal{D}|\theta)$, $p'(\mathcal{D}'|\mathcal{D}, \mathcal{S}', \theta)$ are identical to $p(\theta)$, $p(\mathcal{D}|\theta)$ and $p(\mathcal{D}'|\mathcal{D}, \mathcal{S}', \theta)$ respectively, but differ in $p'(\mathcal{S}|\mathcal{D}, \theta)$:

$$p'(\mathcal{S}|\mathcal{D}, \theta) = \begin{cases} 1 & \text{if } \mathcal{S}' = \mathcal{S}, \\ 0 & \text{else.} \end{cases}$$

That is, p' is identical to p but it assumes that the value of \mathcal{S} is fixed to \mathcal{S}' given \mathcal{D} and θ . For p' , the simulated data \mathcal{S}' is non-informative:

$$-\log_2 p(\mathcal{S}'|\mathcal{D}, \theta) = 0.$$

If one computes the posterior pdf $p'(\theta|\mathcal{D}, \mathcal{S}', \mathcal{D}')$, one obtains the result of Equation 7:

$$\begin{aligned} & \frac{p'(\mathcal{D}'|\mathcal{D}, \mathcal{S}', \theta)p'(\mathcal{S}'|\mathcal{D}, \theta)p'(\mathcal{D}|\theta)p'(\theta)}{\int p'(\mathcal{D}'|\mathcal{D}, \mathcal{S}', \theta')p'(\mathcal{S}'|\mathcal{D}, \theta')p'(\mathcal{D}|\theta')p'(\theta') d\theta'} \\ &= \frac{p(\mathcal{D}'|\mathcal{D}, \mathcal{S}', \theta)p(\mathcal{D}|\theta)p(\theta)}{\int p(\mathcal{D}'|\mathcal{D}, \mathcal{S}', \theta')p(\mathcal{D}|\theta')p(\theta') d\theta'}. \end{aligned}$$

Thus, in order to explain Equation 7 as a posterior pdf given the data sets \mathcal{D} , \mathcal{D}' and the simulated data \mathcal{S}' , one has to *intervene* p in order to account for the fact that \mathcal{S}' is *non-informative given \mathcal{D} and θ* .

In statistics, there is a rich literature on causal intervention. In particular, we will use the formalism developed by Pearl [2000], because it suits the needs to formalize interactions in systems and has a convenient notation—compare Figures 1a & b. Given a *causal model*¹ variables that are intervened are denoted by a hat as in $\hat{\mathcal{S}}$. In the previous example, the causal model of the joint pdf $p(\theta, \mathcal{D}, \mathcal{S}, \mathcal{D}')$ is given by the set of conditional pdfs

$$\mathcal{C}_p = \{p(\theta), p(\mathcal{D}|\theta), p(\mathcal{S}|\mathcal{D}, \theta), p(\mathcal{D}'|\mathcal{D}, \mathcal{S}, \theta)\}.$$

If \mathcal{D} and \mathcal{D}' are observed from the source and \mathcal{S} is intervened to take on the value \mathcal{S}' , then the posterior pdf over the parameters θ is given by $p(\theta|\mathcal{D}, \hat{\mathcal{S}}, \mathcal{D}')$ which is just

$$\begin{aligned} & \frac{p(\mathcal{D}'|\mathcal{D}, \hat{\mathcal{S}}, \theta)p(\hat{\mathcal{S}}|\mathcal{D}, \theta)p(\mathcal{D}|\theta)p(\theta)}{\int p(\mathcal{D}'|\mathcal{D}, \hat{\mathcal{S}}, \theta')p(\hat{\mathcal{S}}|\mathcal{D}, \theta')p(\mathcal{D}|\theta')p(\theta') d\theta'} \\ &= \frac{p(\mathcal{D}'|\mathcal{D}, \mathcal{S}', \theta)p(\mathcal{D}|\theta)p(\theta)}{\int p(\mathcal{D}'|\mathcal{D}, \mathcal{S}', \theta')p(\mathcal{D}|\theta')p(\theta') d\theta'}. \end{aligned}$$

¹For our needs, it is enough to think about a causal model as a complete factorization of a probability distribution into conditional probability distributions representing the causal structure.

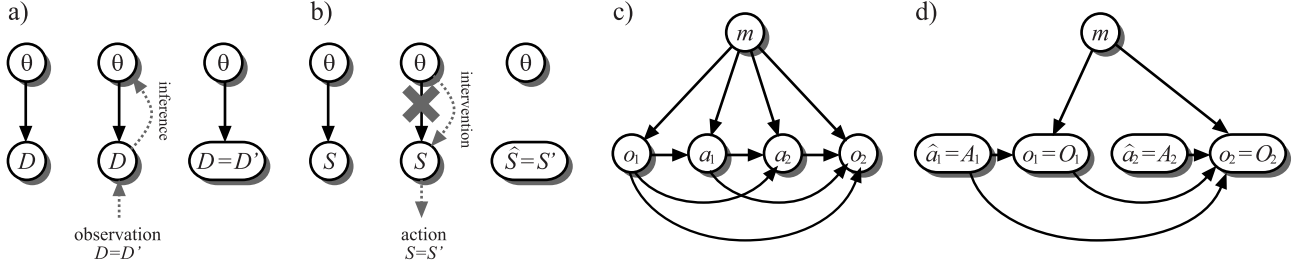


Figure 1: (a-b) Two causal networks, and the result of conditioning on $D = D'$ and intervening on $S = S'$. Unlike the condition, the intervention is set endogenously, thus removing the link to the parent θ . (c-d) A causal network representation of an I/O system with four variables $a_1 o_1 a_2 o_2$ and latent variable m . (c) The initial, un-intervened network. (d) The intervened network after experiencing $\hat{a}_1 o_1 \hat{a}_2 o_2$.

because $p(\mathcal{D}'|\mathcal{D}, \hat{S}', \theta) = p(\mathcal{D}'|\mathcal{D}, S', \theta)$, which corresponds to applying rule 2 in Pearl's intervention calculus, and because $p(\hat{S}'|\mathcal{D}, \theta') = p'(S'|\mathcal{D}, \theta') = 1$.

Adaptive Systems: Causal Construction

Following the discussion in the previous section, we want to construct an adaptive agent \mathbf{P} by minimizing the KL-divergence to the \mathbf{P}_m , but this time treating actions as interventions. Based on the definition of the conditional probabilities in Equation 5, we construct now the KL-divergence criterion to characterize \mathbf{P} using intervention calculus. Importantly, interventions index a set of intervened probability distribution derived from an initial probability distribution. Hence, the set of fixed intervention sequences of the form $\hat{a}_1 \hat{a}_2 \dots$ indexes probability distributions over observation sequences $o_1 o_2 \dots$. Because of this, we are going to construct a set of criteria indexed by the intervention sequences, but we will see that they all have the same solution. Define the history-dependent intervened KL-divergences over the action a_t and observation o_t as

$$C_m^{a_t}(\hat{a}_{o_{<t}}) \equiv \sum_{a_t} P(a_t|m, \hat{a}_{o_{<t}}) \log_2 \frac{P(a_t|m, \hat{a}_{o_{<t}})}{\mathbf{Pr}(a_t|\hat{a}_{o_{<t}})}$$

$$C_m^{o_t}(\hat{a}_{o_{<t}} \hat{a}_t) \equiv \sum_{o_t} P(o_t|m, \hat{a}_{o_{<t}} \hat{a}_t) \log_2 \frac{P(o_t|m, \hat{a}_{o_{<t}} \hat{a}_t)}{\mathbf{Pr}(o_t|\hat{a}_{o_{<t}} \hat{a}_t)},$$

where \mathbf{Pr} is a given arbitrary agent. Note that past actions are treated as interventions. Then, define the average KL-divergences over a_t and o_t as

$$C_m^{a_t} = \sum_{\hat{a}_{o_{<t}}} P(\hat{a}_{o_{<t}}|m) C_m^{a_t}(\hat{a}_{o_{<t}})$$

$$C_m^{o_t} = \sum_{\hat{a}_{o_{<t}} \hat{a}_t} P(\hat{a}_{o_{<t}} \hat{a}_t|m) C_m^{o_t}(\hat{a}_{o_{<t}} \hat{a}_t).$$

Finally, we define the total expected KL-divergence of \mathbf{P} to \mathbf{P}_m as

$$C \equiv \limsup_{t \rightarrow \infty} \sum_m P(m) \sum_{\tau=1}^t (C_m^{a_\tau} + C_m^{o_\tau}). \quad (8)$$

We construct the agent \mathbf{P} as the system that minimizes $C = C(\mathbf{Pr})$:

$$\mathbf{P} \equiv \arg \min_{\mathbf{Pr}} C(\mathbf{Pr}). \quad (9)$$

The solution to Equation 9 is the system \mathbf{P} defined by the set of equations

$$\mathbf{P}(a_t|\hat{a}_{o_{<t}}) = P(a_t|\hat{a}_{o_{<t}})$$

$$= \sum_m P(a_t|m, \hat{a}_{o_{<t}}) v_m(\hat{a}_{o_{<t}})$$

$$\mathbf{P}(o_t|\hat{a}_{o_{<t}} \hat{a}_t) = P(o_t|\hat{a}_{o_{<t}} \hat{a}_t)$$

$$= \sum_m P(o_t|m, \hat{a}_{o_{<t}} \hat{a}_t) v_m(\hat{a}_{o_{<t}} \hat{a}_t) \quad (10)$$

valid for all $\hat{a}_{o_{<t}} \in \mathcal{Z}^*$, where the mixture weights are

$$v_m(\hat{a}_{o_{<t}} \hat{a}_t) = v_m(\hat{a}_{o_{<t}}) \equiv \frac{P(m)P(\hat{a}_{o_{<t}}|m)}{\sum_{m'} P(m')P(\hat{a}_{o_{<t}}|m)}$$

$$= \frac{P(m) \prod_{\tau=1}^{t-1} P(o_\tau|m, \hat{a}_{o_{<\tau}} \hat{a}_\tau)}{\sum_{m'} P(m') \prod_{\tau=1}^{t-1} P(o_\tau|m', \hat{a}_{o_{<\tau}} \hat{a}_\tau)}. \quad (11)$$

The proof follows the same line of argument as the solution to Equation 2 with the crucial difference that actions are treated as interventions. Consider without loss of generality the summand $\sum_m P(m) C_m^{a_t}$ in Equation 8. Note that the KL-divergence can be written as a difference of two logarithms, where only one term depends on \mathbf{Pr} that we want to vary. Therefore, we can integrate out the other term and write it as a constant c . Then we get

$$c - \sum_m P(m) \sum_{\hat{a}_{o_{<t}}} P(\hat{a}_{o_{<t}}|m)$$

$$\cdot \sum_{a_t} P(a_t|m, \hat{a}_{o_{<t}}) \ln \mathbf{Pr}(a_t|\hat{a}_{o_{<t}}).$$

Substituting $P(\hat{a}_{o_{<t}}|m)$ by $P(m|\hat{a}_{o_{<t}})P(\hat{a}_{o_{<t}})/P(m)$ and identifying \mathbf{P} characterized by Equations 10 and 11 we obtain

$$c - \sum_{\hat{a}_{o_{<t}}} P(\hat{a}_{o_{<t}}) \sum_{a_t} \mathbf{P}(a_t|\hat{a}_{o_{<t}}) \ln \mathbf{Pr}(a_t|\hat{a}_{o_{<t}}).$$

The inner sum has the form $-\sum_x p(x) \ln q(x)$, i.e. the cross-entropy between $q(x)$ and $p(x)$, which is minimized when $q(x) = p(x)$ for all x . By choosing this optimum one obtains $\mathbf{Pr}(a_t|\hat{\underline{a}}_{<t}) = \mathbf{P}(a_t|\hat{\underline{a}}_{<t})$ for all a_t . Note that the solution to this variational problem is independent of the weighting $P(\hat{\underline{a}}_{<t})$. Since the same argument applies to any summand $\sum_m P(m)C_m^{a_\tau}$ and $\sum_m P(m)C_m^{o_\tau}$ in Equation 8, their variational problems are mutually independent.

The behavior of \mathbf{P} differs in an important aspect from $\tilde{\mathbf{P}}$. At any given time t , \mathbf{P} maintains a mixture over systems \mathbf{P}_m . The weighting over these systems is given by the mixture coefficients v_m . In contrast to $\tilde{\mathbf{P}}$, \mathbf{P} updates the weights v_m *only* whenever a new observation o_t is produced by the environment respectively. The update follows Bayes' rule but treating past actions as interventions, i.e. dropping the evidence they provide. In addition, \mathbf{P} issues an action a_t suggested by an system m drawn randomly according to the weights v_m —see Figures 1c & d.

If we use the following equalities connecting the weights and the intervened posterior distributions

$$v_m(\underline{a}o_{<t}) = P(m|\hat{\underline{a}}o_{<t}) = P(m|\hat{\underline{a}}o_{<t}\hat{a}_t) = v_m(\underline{a}o_{<t}a_t)$$

and substitute interventions by observations in the conditionals

$$\begin{aligned} P(a_t|m, \hat{\underline{a}}o_{<t}) &= P(a_t|m, \underline{a}o_{<t}) \\ P(o_t|m, \hat{\underline{a}}o_{<t}\hat{a}_t) &= P(o_t|m, \underline{a}o_{<t}a_t) \end{aligned}$$

which corresponds to rule 2 of Pearl's intervention calculus, we can rewrite Equations 10 and 11 as

$$\begin{aligned} \mathbf{P}(a_t|\underline{a}o_{<t}) &= P(a_t|\hat{\underline{a}}o_{<t}) \\ &= \sum_m P(a_t|m, \underline{a}o_{<t})P(m|\hat{\underline{a}}o_{<t}) \quad (12) \\ \mathbf{P}(o_t|\underline{a}o_{<t}a_t) &= P(o_t|\hat{\underline{a}}o_{<t}\hat{a}_t) \\ &= \sum_m P(o_t|m, \underline{a}o_{<t}a_t)P(m|\hat{\underline{a}}o_{<t}) \quad (13) \end{aligned}$$

where the intervened posterior probabilities are

$$P(m|\hat{\underline{a}}o_{<t}) = \frac{P(m) \prod_{\tau=1}^{t-1} P(o_\tau|m, \underline{a}o_{<\tau}a_\tau)}{\sum_{m'} P(m') \prod_{\tau=1}^{t-1} P(o_\tau|m', \underline{a}o_{<\tau}a_\tau)}. \quad (14)$$

Equations 12, 13 and 14 are important because they describe the behavior of \mathbf{P} only in terms of known probabilities, i.e. probabilities that are computable from the causal model associated to P given by

$$C_P = \{P(m), P(a_t|m, \underline{a}o_{<t}), P(o_t|m, \underline{a}o_{<t}a_t) : t \geq 1\}.$$

Importantly, Equation 12 describes a stochastic method to produce desirable actions that differs fundamentally from an agent that is constructed by choosing an optimal policy with respect to a given utility criterion. We call this action selection rule the *Bayesian control rule*.

Experimental Results

Here we design a very simple toy experiment to illustrate the behavior of an agent $\tilde{\mathbf{P}}$ based on a Bayesian mixture compared to an agent \mathbf{P} based on the Bayesian control rule.

Let $\mathbf{Q}_0, \mathbf{Q}_1, \mathbf{P}_0$ and \mathbf{P}_1 be four agents with binary I/O sets $\mathcal{A} = \mathcal{O} = \{0, 1\}$ defined as follows. \mathbf{P}_1 is such that $\mathbf{P}_1(a_t|\underline{a}o_{<t}) = \mathbf{P}_1(a_t)$ and $\mathbf{P}_1(o_t|\underline{a}o_{<t}a_t) = \mathbf{P}_1(o_t)$ for all $\underline{a}o_{<t} \in \mathcal{Z}^*$, where

$$\mathbf{P}_1(a_t) = \begin{cases} 0.1 & \text{if } a_t = 0 \\ 0.9 & \text{if } a_t = 1 \end{cases}, \quad \mathbf{P}_1(o_t) = \begin{cases} 0.4 & \text{if } a_t = 0 \\ 0.6 & \text{if } a_t = 1 \end{cases}.$$

Let \mathbf{P}_0 be such that

$$\begin{aligned} \mathbf{P}_0(a_t|\underline{a}o_{<t}) &= 1 - \mathbf{P}_1(a_t|\underline{a}o_{<t}) \\ \mathbf{P}_0(o_t|\underline{a}o_{<t}a_t) &= 1 - \mathbf{P}_0(o_t|\underline{a}o_{<t}a_t) \end{aligned}$$

for all $\underline{a}o_{<t} \in \mathcal{Z}^*$. Thus, \mathbf{P}_0 and \mathbf{P}_1 are agents that are biased towards observing and acting 0's and 1's respectively. Furthermore, $\mathbf{Q}_0 = \mathbf{P}_0$ and $\mathbf{Q}_1 = \mathbf{P}_1$. Assume a uniform distribution over $\mathcal{Q} = \{\mathbf{Q}_0, \mathbf{Q}_1\}$, i.e. $P(m=0) = P(m=1) = \frac{1}{2}$.

Assume $\mathbf{Q}_0 \in \mathcal{Q}$ is drawn. In this case, one wants the agents $\tilde{\mathbf{P}}$ and \mathbf{P} to minimize the deviation from \mathbf{P}_0 . Consider the following instantaneous measure

$$\begin{aligned} d(t) &\equiv \sum_{a'_t} \mathbf{P}_0(a'_t) \log_2 \frac{\mathbf{P}_0(a'_t)}{\mathbf{Pr}(a'_t|\underline{a}o_{<t})} \\ &+ \sum_{o'_t} \mathbf{P}_0(o'_t) \log_2 \frac{\mathbf{P}_0(o'_t)}{\mathbf{Pr}(o'_t|\underline{a}o_{<t}a_t)} \end{aligned}$$

where $a_1 o_1 a_2 o_2 \dots$ is a realization of the interaction system $(\mathbf{Pr}, \mathbf{Q}_0)$. $d(t)$ measures how much \mathbf{Pr} 's action and observation probabilities deviate from \mathbf{P}_0 at time t .

Recall that both $\tilde{\mathbf{P}}$ and \mathbf{P} maintain a mixture over \mathbf{P}_0 and \mathbf{P}_1 . The instantaneous I/O probabilities of such a system can always be written as

$$\begin{aligned} w\mathbf{P}_0(a_t) + (1-w)\mathbf{P}_1(a_t) \\ w\mathbf{P}_0(o_t) + (1-w)\mathbf{P}_1(o_t). \end{aligned}$$

where $w \in [0, 1]$. Thus, it is easy to see that the instantaneous I/O deviation takes on the minimum value when $w = 1$ and the maximum value when $w = 0$: In the case $w = 1$, $d(t) = 0$ bits; In the case $w = 0$, $d(t) \approx 2.653$.

We have simulated realizations of the instantaneous I/O deviation using the agents $\tilde{\mathbf{P}}$ and \mathbf{P} . The results are summarized in Figure 2. For $\tilde{\mathbf{P}}$, $d(t)$ happens to be non-ergodic: it either converges to $d(t) \rightarrow 0$ or to $d(t) \rightarrow \approx 2.654$, implying that either $\tilde{\mathbf{P}} \rightarrow \mathbf{P}_0$ or $\tilde{\mathbf{P}} \rightarrow \mathbf{P}_1$ respectively. In contrast, $d(t) \rightarrow 0$ always for \mathbf{P} , implying that $\mathbf{P} \rightarrow \mathbf{P}_0$.

Analogous results are obtained when $\mathbf{Q}_1 \in \mathcal{Q}$ is drawn instead: For $\tilde{\mathbf{P}}$, $d(t)$ converges either to 0 or to ≈ 2.654 , whereas for \mathbf{P} , $d(t) \rightarrow \approx 2.654$ always implying that $\mathbf{P} \rightarrow \mathbf{P}_1$. Hence, \mathbf{P} shows the correct adaptive behavior while $\tilde{\mathbf{P}}$ does not.

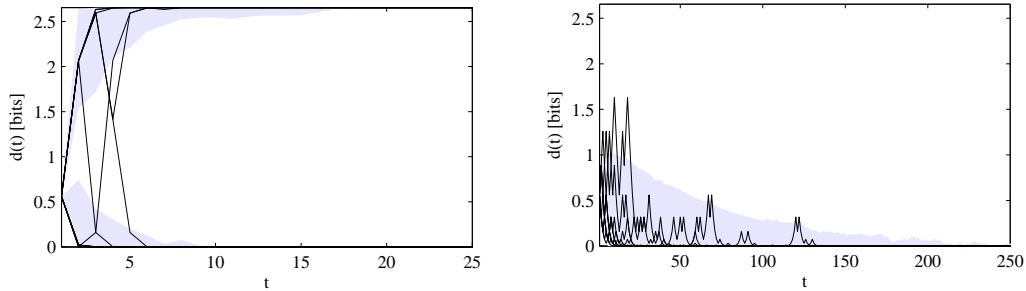


Figure 2: 10 realizations of the instantaneous deviation $d(t)$ for the agents $\tilde{\mathbf{P}}$ (left panel) and \mathbf{P} (right panel). The shaded region represents the standard deviation barriers computed over 1000 realizations. Since $d(t)$ is non-ergodic for $\tilde{\mathbf{P}}$, we have separated the realizations converging to 0 from the realizations converging to ≈ 2.654 to compute the barriers. Note that the time scales differ in one order of magnitude.

Conclusions

We propose a Bayesian rule for adaptive control. The key feature of this rule is the special treatment of actions based on causal calculus and the decomposition of agents into Bayesian mixture of I/O distributions. The question of how to integrate information generated by an agent’s probabilistic model into the agent’s information state lies at the very heart of adaptive agent design. We show that the naïve application of Bayes’ rule to I/O distributions leads to inconsistencies, because outputs don’t provide the same type of information as genuine observations. Crucially, these inconsistencies vanish if intervention calculus is applied [Pearl, 2000].

Some of the presented key ideas are not unique to the Bayesian control rule. The idea of representing agents and environments as I/O streams has been proposed by a number of other approaches, such as predictive state representation (PSR) [Littman et al., 2002] and the universal AI approach by Hutter [2004]. The idea of breaking down a control problem into a superposition of controllers has been previously evoked in the context of “mixture of experts”-models like the MOSAIC-architecture Haruno et al. [2001]. Other stochastic action selection approaches are found in exploration strategies for (PO)MDPs [Wyatt, 1997], learning automata [Narendra and Thathachar, 1974] and in probability matching [R.O. Duda, 2001] amongst others. The usage of compression principles to select actions has been proposed by AI researchers, for example Schmidhuber [2009]. The main contribution of this paper is the derivation of a stochastic action selection and inference rule by minimizing KL-divergences of intervened I/O distributions.

An important potential application of the Bayesian control rule would naturally be the realm of adaptive control problems. Since it takes on a similar form to Bayes’ rule, the adaptive control problem could then be translated into an on-line inference problem where actions are sampled stochastically from a posterior distribution. It is important to note, however, that the problem statement as formulated here and the usual

Bayes-optimal approach in adaptive control are *not* the same. In the future the relationship between these two problem statements deserves further investigation.

References

- R. Beer. *Intelligence as Adaptive Behavior*. Academic Press, Inc., 1990.
- M. Haruno, D.M. Wolpert, and M. Kawato. Mosaic model for sensorimotor learning and control. *Neural Computation*, 13:2201–2220, 2001.
- D. Haussler and M. Opper. Mutual information, metric entropy and cumulative relative entropy risk. *The Annals of Statistics*, 25:2451–2492, 1997.
- M. Hutter. *Universal Artificial Intelligence: Sequential Decisions based on Algorithmic Probability*. Springer, Berlin, 2004.
- M. Littman, R. Sutton, and S. Singh. Predictive representations of state. In *Neural Information Processing Systems (NIPS)*, number 14, pages 1555–1561, 2002.
- D.J.C. MacKay. *Information Theory, Inference, and Learning Algorithms*. Cambridge University Press, 2003.
- K. Narendra and M.A.L. Thathachar. Learning automata - a survey. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-4(4):323–334, July 1974.
- M. Opper. A bayesian approach to online learning. *Online Learning in Neural Networks*, pages 363–378, 1998.
- J. Pearl. *Causality: Models, Reasoning, and Inference*. Cambridge University Press, Cambridge, UK, 2000.
- D.G. Stork R.O. Duda, P.E. Hart. *Pattern Classification*. Wiley & Sons, Inc., second edition, 2001.
- J. Schmidhuber. Simple algorithmic theory of subjective beauty, novelty, surprise, interestingness, attention, curiosity, creativity, art, science, music, jokes. *Journal of SICE*, 48(1):21–32, 2009.
- J. Wyatt. *Exploration and Inference in Learning from Reinforcement*. PhD thesis, Department of Artificial Intelligence, University of Edinburgh, 1997.

Sketch of an AGI architecture with illustration

András Lőrincz and Zoltán R. Bárdosi and Dániel Takács

Eötvös Loránd University
Pázmány Péter s. 1/C, Budapest, Hungary 1117

Abstract

Here we present a framework for AGI inspired by knowledge about the only working prototype: the brain. We consider the neurobiological findings as directives. The main algorithmic modules are defined and solutions for each subtasks are given together with the available mathematical (hard) constraints. The main themes are compressed sensing, factor learning, independent process analysis and low dimensional embedding for optimal state representation to be used by a particular RL system that can be integrated with a robust controller. However, the blending of the suggested partial solutions is not a straightforward task. Nevertheless we start to combine these modules and illustrate their working on a simulated problem. We will discuss the steps needed to complete the integration.

Introduction

In recent years, attempts have been made to understand the algorithmic principles of general intelligence. The first issue of the Cognitive Computation Journal reviews questions, including, e.g., if human like intelligence is achievable [McC09], ‘howto’ do the reverse engineering of the vertebrate brain [Gur09] among others that are relevant for artificial general intelligence (AGI) and neuroscience. Some of the AGI works started early and built upon theories of cognition, like SOAR (for a review, see [Lai09]) and Clarion (see, e.g., [Sun07] and references therein). Recent works consider universal searches [Hut09] and [Sch09] and build universal algorithms.

Another approach tries to limit the number of available algorithms and considers constraints [Lör09b]: *directive soft constraints* come from neuroscience and cognition. These constraints are soft, because the interpretation of the findings is debated in many cases. Another type of constraints comes from mathematics. These are *hard constraints*. However, care should be taken as they may also have their own pitfalls hidden deeply in the conditions of the different theorems applied. The central question to these types of constraints is what algorithms can limit combinatorial explosion. In this paper we elaborate on the control issues of this approach.

The main contributions of the paper is the illustration of (i) how high-dimensional sensory information can be connected to autoregressive exogenous (ARX) processes via low-dimensional embedding, (ii) how one can estimate the inverse dynamics from the ARX model, and (iii) how one can connect the low-dimensional map to the event-learning framework [STL03] of reinforcement learning.

In the next section we review the basic components of the architecture. Then we highlight the working of some of the components through an illustrative example. Short discussion and conclusions about some experimental possibilities close the paper.

Background

We extend the architecture detailed in [Lör09b]. We also consider missing components. We start from the observation that natural data – in many cases, but not always – exhibit heavy-tailed distributions. Such distributions may satisfy the conditions of Compressive Sensing (CS), a highly advantageous feature if part of the information is missing (see Compressive Sensing Resources <http://dsp.rice.edu/cs>).

Since heavy-tailed distribution is not warranted, the processing of sensory information in the architecture utilizes ‘cross entropy’ global probabilistic search [Rub97] to optimize overcomplete sparse representation using L_0 norm [LPS08]. Recent results on CS indicate (see, [DTDS07, NV07] and references therein) and numerical studies reinforce [PL09a] that certain versions of orthogonal matching pursuit complement and speed-up the cross entropy method.

One can alleviate the problem of combinatorial explosion by separating information of different kinds into (quasi-) independent lower-dimensional subspaces. As an example, facial speech and expressions can be represented in lower dimensions, respectively, by applying bilinear factorization [CDB02]. Since sparse codes and independent component analysis are related to each other [OF97], one option is that multi-linear extensions of Independent Process Analysis (IPA) [SL09a], similar to the multi-linear extension of Independent Component Analysis [VT05] may solve the separation problem. It is reassuring soft information from neuroscience that

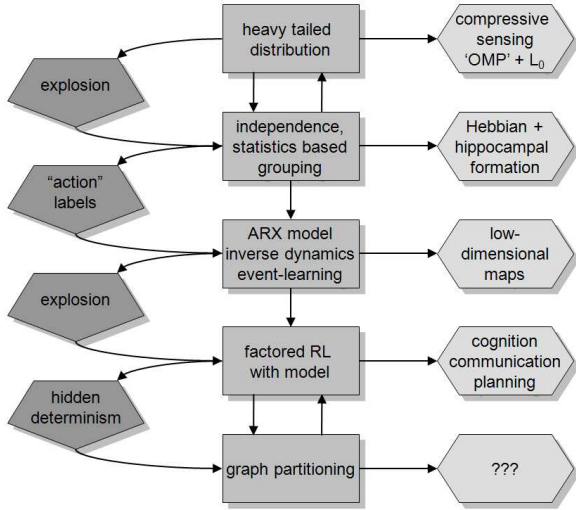


Figure 1: Squares: algorithmic components. Pentagons: emergent problems that need to be solved to proceed further. Hexagons: reassuring soft information from neuroscience and cognition, ‘OMP’: some version of orthogonal matching pursuit, L_0 : probabilistic sparsification using L_0 norm.

the constraint of *Hebbian learning* on IPA gives rise to an architecture that closely resembles the entorhinal-hippocampal loop [Lör09a, LS09a] that plays a central role in the encoding of episodic memory, or *events* in the brain.

There are a number of missing parts before this portion of the AGI architecture could be built, such as (i) the method to fuse information from different information sources (or modalities), (ii) the method to learn and embed the independent factors into different low-dimensional networks, and (iii) the method to extend the linear recurrent IPA network to non-linear ones (for a comprehensive lists on recurrent networks, see <http://www.idsia.ch/~juergen/rnn.html> and [LH09]).

From the point of view of controlling, IPA separates the relevant sensory information, such as position and direction [LS09a]. Furthermore, one can learn the effect of control – in an exogenous autoregressive (ARX) process – by optimal experimental design both for the observed [PL09b] and for the hidden variables [SL09a]. Inversion of the learned relations can be used as estimates of the inverse dynamics: it can derive control values for a desired state given the actual (experienced) one. This feature is advantageous for our architecture that applies robust controllers [LHS01].

Approximate inverse dynamics is a necessary pre-supposition of *event-learning* [STL03] that works with a background controller and optimizes policy with regard to desired (planned) states. We note that event learning admits robust controllers.

Combinatorial explosion, however, spoils event-learning, unless factors and thus factored RL methods are available. It has been shown that (i) factored methods combined with sampling converge in polynomial time [SL08a], (ii) the exploration–exploitation dilemma can be overcome for optimistic initial models even for factored RL [SL09b], and (iii) the RL optimization remains polynomial.

We list two problems that need to be solved in this architecture:

Factor learning. It is unclear how to learn the factors in general. One suggestion considers factors as the primitives of symbols and the ‘*symbol learning problem*’ as structure-noise separation of graphs [Lör09b]. The corresponding graph partitioning problem is polynomial even for extreme graphs [Tao06]

Control of an under-controlled plant. The robust controller assumes an over-controlled plant (i.e., the number of controls is larger than the number of freedom). This condition may not be satisfied in general.

In what follows, we illustrate how to solve the last point. We shall start from high-dimensional space, will embed it into a low-dimensional one, but the control will remain undercomplete. We will use optimal control methods to overcome this obstacle.

Architecture for the under-controlled case

We present a simplified version of the architecture described in [Lör09b]. We treat the following stages:

Sample selection: This stage selects samples under random control. Selected states are not too close to each other.

Low-dimensional embedding: In our illustration, the dimension of the low dimensional manifold is known, so we do not have to estimate it. Selected samples are embedded into the low dimensional space.

Identification of the ARX process: Out-of-sample estimations in the low-dimensional space are used for the identification of the ARX process.

LQR solution to the inverse dynamics: We have an under-controlled situation. We use a linear-quadratic regulator (LQR) to overcome this problem.

Exploring the space: Optimistic initial model (OIM) is used for exploring the space and for learning the values of different initial and final state-pairs, or *events*.

Now, we introduce the illustration.

Illustration with a pendulum

In the early phase of learning, the pendulum was subject to random control:

$$\ddot{\psi} = -\frac{g}{l}\sin(\psi) - \frac{\gamma\dot{\psi}}{ml^2} + \frac{f}{ml^2}$$

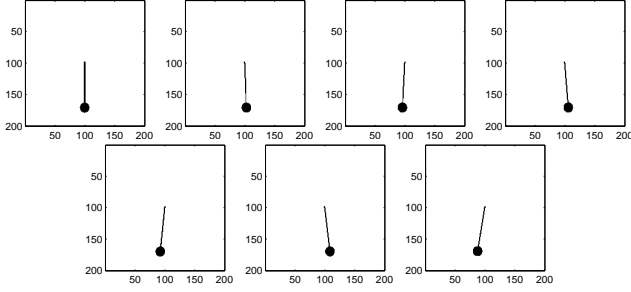


Figure 2: Selected sensors.

where ψ is the angle of the pendulum, and parameters were set as $g = 9.81 \frac{m}{s^2}$, $m = 1kg$, $l = 100m$, and $\gamma = 500$. We limited the value of control f either to $100N$, or to $200N$ in our experiments.

Sample selection

We used 200×200 images of a pendulum (Fig. 2), so the input dimension was 40,000. In order to lower the dimension, we choose only a few samples (our sensors) from the input space. For similarity estimation, we used a Gaussian kernel: $K(x, y) = \exp\left(-\frac{\|x-y\|^2}{\sigma^2}\right)$ with $\sigma = 0,75$. For a set of sensors $S = \{s_1, \dots, s_{|S|}\}$ the sensor space is determined by this kernel: for any input x , the sensed values form a vector in $\mathbb{R}^{|S|}$

$$\phi_S(x) = (K(s_1, x), \dots, K(s_{|S|}, x))^T.$$

During sample generation we used $f_{max}^{rnd} = 100N$ and limited by time

Algorithm 1 Time limited sample generation

Given: duration of training T , initial angle ψ_0
 $img_0 \leftarrow genPendImage(a_0)$
for $t = 0 \dots T-1$ **do**
 $f_t \leftarrow genRandControl()$
 $\psi_{t+1} \leftarrow simulatePendulum(f_t)$
 $img_{t+1} \leftarrow genPendImage(\psi_{t+1})$
end for

Algorithm 2 Similarity based sensor selection

$S \leftarrow \{img_0\}$
for $t = 1 \dots T$ **do**
if $\forall s \in S : K(s, img_t) < 0.3$ **then**
 $S \leftarrow S \cup \{img_t\}$
end if
end for

After time limited sensor selection, low dimensional embedding takes place.

Low-dimensional embedding and out-of-sample estimation

We used different input sets, including the set of simple views of the pendulum. We have tried a number of embedding algorithms and found that ISOMAP [BST⁺02] suits our input sets the best. On the selected data we applied the ISOMAP embedding algorithm with 3 nearest neighbors and generated a 1-dimensional embedding.

For inputs, which were not included into set S we approximated the corresponding 1-dimensional value by means of the partial least square (PLS) regression method (for a review and the extensions of the method, see [RK06]). The procedure is summarized below:

Algorithm 3 Embedding and out-of-sample estimation

{Data to be embedded:}
 $A \leftarrow \emptyset$
for all $s \in S$ **do**
 $A \leftarrow A \cup \{\phi_S(s)\}$
end for

 {Embedding:}
 $G \leftarrow ISOMAP(A, 1D, 3-NN)$
 {Then $G \subset \mathbb{R}, G = \{g_a | a \in A\}$ }

 {Out-of-state estimations}
for $t = 0 \dots T$ **do**
 $\phi_t \leftarrow \phi_S(img_t)$
 $N \leftarrow SelectNearest(A, \phi_t, 3)$
 $\hat{x}_t \leftarrow PLS(N, \{g_a \in G | a \in N\}, \phi_t)$
end for

Results are shown in Fig. 3

Identification of the ARX process

The following equation of motion has been assumed

$$x_{t+1} = A_1 x_t + A_2 x_{t-1} + B u_t + \varepsilon_t \quad (1)$$

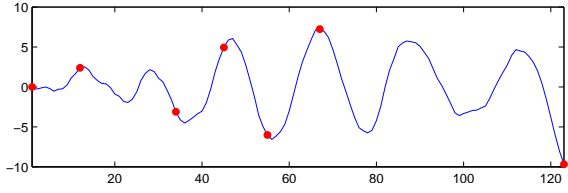
where $A_1, A_2, B \in \mathbb{R}$, ε is a normally distributed stochastic variable with 0 mean and covariance σ . Note, however, that this assumption may not hold; it is subject to our estimation errors, including the error of the PLS estimation. We have estimated parameters B, A_1 and A_2 with a least squares estimation derived from the cost function:

$$J(A_1, A_2, B) = \sum_{t=1}^T (\hat{x}_{t+1} - A_1 \hat{x}_t - A_2 \hat{x}_{t-1} - B u_t)^2$$

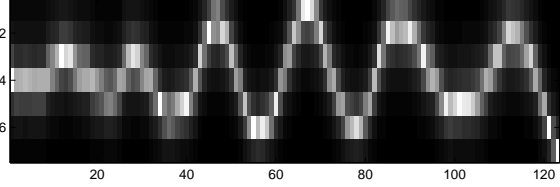
where \hat{x} denotes the PLS estimated coordinates. We also tried an on-line Bayesian method that gave similar results.

LQR solution to the inverse dynamics

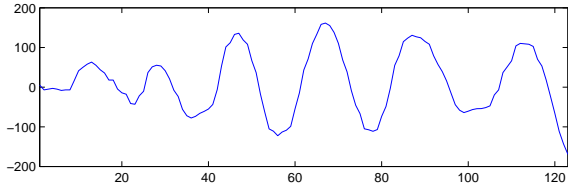
We define the control problem by means of *event learning* RL algorithm [STL03] that provides the actual and



(a) Motion of the pendulum as a function of time in angle space between $\pm 10^\circ$. Motion is subject to random control. Red dots: positions of selected sensors



(b) Responses in sensor space. The outputs of the sensors are shown vs. time. Sensors are ordered according to their 1-dimensional embeddings.



(c) PLS position estimation in the embedded 1-dimensional space during the motion of the pendulum

Figure 3: Motion of the pendulum in real space, in sensor space and in embedded 1-dimensional space.

desired states to a backing controller. The controller tries to satisfy ‘desires’. For given experienced state and desired state pair the controller provides a control value or a control series. Then, event learning learns the limitations of the backing controller and optimizes the RL policy in the event space accordingly.

In the present scenario, we have a 1-dimensional plant of second order, so the state of the plant is determined by its position and its speed, or alternatively, by its present and previous positions. The same holds for the desired state, which thus has two degrees of freedom. For our 1D control problem, 2 time steps are needed to reach the desired state, if it is possible at all. If it is, then there are many solutions, out of which an optimal control can help to choose: one assumes costs associated with the trajectory and the value of the control. Denoting the state by $q_t = (x_t, x_{t-1})^T$, we can convert Eq. 1 into the following form $q_{t+1} = Fq_t + Cv_t + \epsilon_t$, where

$$F = \begin{bmatrix} A_1 & A_2 \\ 0 & I \end{bmatrix} \quad (2)$$

$C = (B, 0)^T$, $v_t = (u_t, 0)^T$, and $\epsilon_t = (\epsilon_t, 0)^T$.

For N -step experienced state q_i and desired state q_i^d ($i = 1, \dots, N$) one may use quadratic cost functions both

for the $q_i - q_i^d$ differences and for the control values using the cost function

$$J(U) = \sum_{i=0}^N (q_i - q_i^d)^T Q_i (q_i - q_i^d) + \sum_{i=0}^{N-1} v_i^T R_i v_i \quad (3)$$

where $Q_i = Q_i^T \geq 0$ and similarly, $R_i = R_i^T \geq 0$. Then,

$$\begin{bmatrix} q_1 \\ \vdots \\ q_N \end{bmatrix} = H \begin{bmatrix} v_0 \\ \vdots \\ v_{N-1} \end{bmatrix} + \begin{bmatrix} I \\ \vdots \\ F^N \end{bmatrix} q_0$$

where

$$H = \begin{bmatrix} 0 & \dots & 0 & \dots & 0 \\ C & 0 & C & 0 & \dots \\ FC & C & 0 & \dots & \dots \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ F^{N-1}C & F^{N-2}C & \dots & C & \dots \end{bmatrix}$$

that we can write in the following short form: $\mathbf{q} = H\mathbf{v} + Gq_0$, where $\mathbf{q} = (q_0, \dots, q_N)^T$, $\mathbf{v} = (v_0, \dots, v_N)^T$, and $G = (I, \dots, F^N)^T$. Now, the cost function can be rewritten into the following quadratic form

$$J(U) = \|\text{diag}(Q_0^{\frac{1}{2}}, \dots, Q_N^{\frac{1}{2}})(H\mathbf{v} + Gq_0 - \mathbf{q})\|^2 + \|\text{diag}(R_0^{\frac{1}{2}}, \dots, R_{N-1}^{\frac{1}{2}})\mathbf{v}\|^2 \quad (4)$$

that can be solved by simple routines. In our case, $Q_0, \dots, Q_{N-2} = 0$, $Q_{N-1}, Q_N = 1$, and $R_0 \dots R_{N-1} = R$.

Now, we turn to the optimization of the RL problem.

Exploring the space: experienced transitions

The estimated value of an event $E_{i,j}^\pi$ in event learning is the estimated long-term cumulated discounted reward of event $e = (q_i, q_j^+)$ in a Markov decision process under fixed policy $\pi = \pi(q, q^+)$, with actual state q and desired state q^+ . The plus sign indicates that the second argument of event e is the successor state of the state represented by the first argument. We note that the OIM policy [SL08b] is not fixed, but it converges to the optimal policy. Here we use OIM for the exploration of the space. OIM, however, could be used for diverse tasks, e.g., for inverting the pendulum, like in [STL03].

We used $f_{max}^{OIM} = 200N$ in this case to allow OIM to connect a larger number of states. Experimental results are shown in Figs. 4 and 5.

Discussion and conclusions

The presented toy problem and its solution highlight the following issues relevant to AGI

- Factors like position, speed, etc. can be learned by neural mechanisms [LS09b]. On the other hand, we do not know how to learn factors in general. Multi-linear IPA might work for some cases.

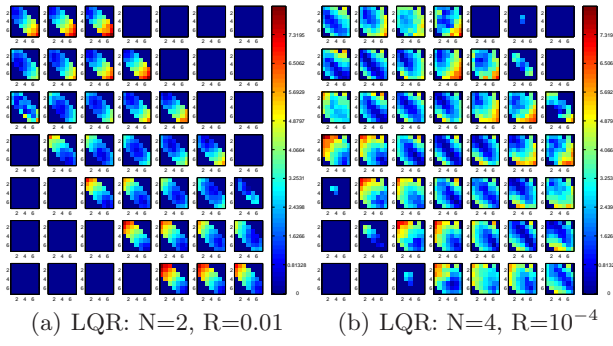


Figure 4: Color coded error between desired and experienced states using OIM. Each block represents one of the $7 \times 7 = 49$ actual states. The horizontal and vertical indices of the subfigures represent the number of the actual position and the number of the previous position, respectively. Each square within the each block represents a desired state and is also indexed according to the actual and the previous positions. The warmer the color the error. Transitions corresponding to large blue areas have not occurred during the course of the experiment. The larger the number of time steps used by LQR and the smaller the cost of the control, the more states can be reached. OIM addresses the exploration-exploitation dilemma by estimating the transition probabilities and thus enabling planning.

- We suspect factors correspond to a highly simplified directed bipartite graph derived from a highly complex graph connecting the products of observed actual and desired states to the next state. Weights of this graph represent the transition probabilities. According to the conjecture [Lőr09b], one can use the separated structured part of the graph. The vertices of this graph make the factors. The rest of the original graph is made of noisy Erdős-Rényi-like blocks and the structured part saves the transition probabilities. It has been shown that such compression is possible even for extreme graphs [Tao06].
- We used low-dimensional embedding – suggested by neuronal control systems, like the ‘gelatinous medium’ surrounding the limb [GG93] – in order to transform high-dimensional information for the sake of lower dimensional and possibly robust control.
- Optimal control can serve the optimization of under-controlled plants. Beyond the presented LQR method, which assumes no noise, extensions to observations spoiled by Gaussian noise are available [SL04]. Further extensions to stochastic optimal control are desired.
- ARX based *optimal control*, such as LQR control, has feedforward characteristics. On the other hand, *robust control* works by error correction. *Experienced events* that emerge through the application of these controllers or their combinations require further op-

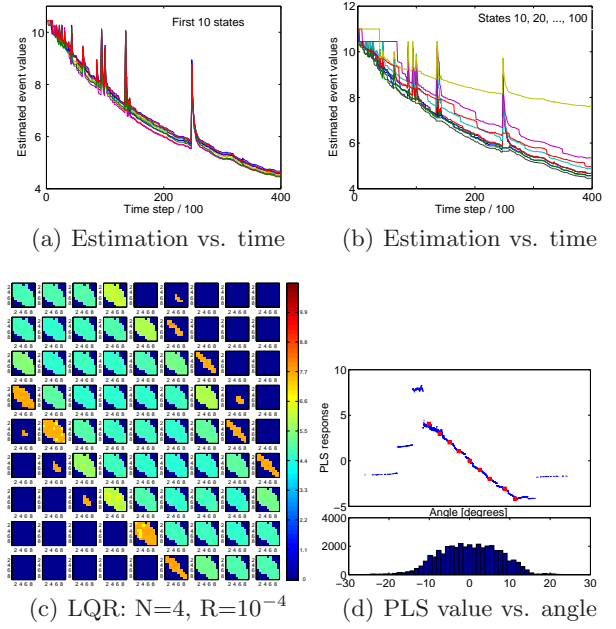


Figure 5: Convergence of event values $E_{i,j}$ (see text). (a): Convergence of the OIM estimation for the first 10 events. (b): Convergence of every 10^{th} event from the first 100 events. (c): Estimated values of all found events 40,000 steps. The warmer the color, the larger the OIM estimated value. Transitions corresponding to large blue areas have not occurred until the experiment was finished. (d): Out-of-sample estimation vs. pendulum angle and histogram of positions in arbitrary units after 40,000 steps. Spikes and sudden drops correspond to the discoveries of new states and to visits to known states, respectively.

timization using *RL based optimal control*.

From the point of view of neuroscience, the roles of the two control related networks, the cerebellum and the basal ganglia are controversial [Bas06, YBK09]. We suspect that this controversy arises from the fact that optimal control appears in two ways: once for feedforward control of plants, and once for optimal decision making. We conjecture that one might gain more insight into the functioning of these neural architectures by comparing their roles in over-controlled and under-controlled control tasks.

Acknowledgements

We are most grateful to Zoltán Szabó for helpful discussions and to Gábor Szirtes for careful reading of the manuscript. This research has been partially supported by the EC NEST ‘Percept’ grant under contract 043261. Opinions and errors in this manuscript are the author’s responsibility, they do not necessarily reflect the opinions of the EC or other project members.

References

- [Bas06] A. J. Bastian. Learning to predict the future: the cerebellum adapts feedforward movement control. *Current Opinion in Neurobiology*, 16:645–649, 2006.
- [BST⁺02] M. Balasubramanian, E. L. Schwartz, J. B. Tenenbaum, V. de Silva, and J. C. Langford. The ISOMAP algorithm and topological stability. *Science*, 290:2319–2323, 2002.
- [CDB02] E. S. Chuang, H. Deshpande, and C. Bregler. Facial expression space learning. In *Proc. Pacific Conf. Comp. Graphics Appl.*, 2002. <http://cims.nyu.edu/~bregler/pubs.html>.
- [DTDS07] D. L. Donoho, Y. Tsaig, I. Drori, and J.-L. Starck. Sparse solution of underdetermined linear equations by stagewise orthogonal matching pursuit. Technical Report, 2007. <http://stat.stanford.edu/~idrori/StOMP.pdf>.
- [GG93] M. S. Graziano and C. G. Gross. A bimodal map of space: somatosensory receptive fields in the macaque putamen with corresponding visual receptive fields. *Experimental Brain Research*, 97:96–109, 1993.
- [Gur09] K. N. Gurney. Reverse engineering the vertebrate brain. *Cogn. Comput.*, 1:29–41, 2009.
- [Hut09] M. Hutter. Feature reinforcement learning: Part I. Unstructured MDPs. *J. of AGI*, 1:3–24, 2009.
- [Lai09] J. Laird. Soar cognitive architecture tutorial. In *2nd Conf. on Artificial General Intelligence*, AGI-2009, 2009. <http://agi-conf.org/2009/slides/>.
- [LH09] M. Lukosevicius and J. Herbert. Reservoir computing approaches to recurrent neural network training. *Comp. Sci. Rev.*, 3:127–149, 2009.
- [LHS01] A. Lőrincz, Gy. Hévízi, and Cs. Szepesvári. Ockham’s razor modeling of the matrisome channels of the basal ganglia thalamocortical loop. *Int. J. of Neural Syst.*, 11:125–143, 2001.
- [Lőr09a] A. Lőrincz. Hebbian constraint on the resolution of the homunculus fallacy leads to a network that searches for hidden cause-effect relationships. In *AGI-2009*, volume 8 of *Adv. in Intell. Syst. Res. (ISSN: 1951-6851)*, pages 126–131. Atlantis Press, 2009.
- [Lőr09b] A. Lőrincz. Learning and representation: From compressive sampling to the ‘symbol learning problem’. In *Handbook of Large-Scale Random Networks*, pages 445–488. Springer, Germany, 2009.
- [LPS08] A. Lőrincz, Zs. Palotai, and G. Szirtes. Spike-based cross-entropy method for reconstruction. *Neurocomputing*, 71:3635–3639, 2008.
- [LS09a] A. Lőrincz and G. Szirtes. Here and now: how time segments may become events in the hippocampus. *Neural Networks*, 22:738–747, 2009.
- [LS09b] A. Lőrincz and G. Szirtes. Representation theory meets anatomy: Factor learning in the hippocampal formation. In *Connectionist Models of Behaviour and Cognition II*, volume 18, pages 253–264, Singapore, 2009. World Scientific.
- [McC09] J. L. McClelland. Is a machine realization of truly human-like intelligence achievable? *Cogn. Comput.*, 1:17–21, 2009.
- [NV07] D. Needell and R. Vershynin. Signal recovery from incomplete and inaccurate measurements via regularized orthogonal matching pursuit. 2007. <http://arxiv.org/abs/0712.1360>.
- [OF97] B. A. Olshausen and D. J. Field. Sparse coding with an overcomplete basis set: A strategy employed by V1? *Vision Research*, 37:3311–3325, 1997.
- [PL09a] Zs. Palotai and A. Lőrincz. OMP speeds up CE in L_0 norm optimizations. unpublished, 2009.
- [PL09b] B. Póczos and A. Lőrincz. Identification of recurrent neural networks by Bayesian interrogation techniques. *J. of Mach. Learn. Res.*, 10:515–554, 2009.
- [RK06] R. Rosipal and N. Kramer. *Subspace, Latent Structure and Feature Selection Techniques*, chapter Overview and Recent Advances in Partial Least Squares, pages 34–51. Springer, 2006.
- [Rub97] R. Y. Rubinstein. Optimization of computer simulation models with rare events. *European Journal of Operations Research*, 99:89–112, 1997.
- [Sch09] J. Schmidhuber. Ultimate cognition á la Gödel. *Cogn. Comput.*, 1:177–193, 2009.
- [SL04] I. Szita and A. Lőrincz. Kalman filter control embedded into the reinforcement learning framework. *Neural Comp.*, 16:491–499, 2004.
- [SL08a] I. Szita and A. Lőrincz. Factored value iteration converges. *Acta Cyb.*, 18:615–635, 2008.
- [SL08b] I. Szita and A. Lőrincz. The many faces of optimism: a unifying approach. In *ICML 2008*, pages 1048–1055, Helsinki, 2008. Omnipress.
- [SL09a] Z. Szabó and A. Lőrincz. Controlled complete ARMA independent process analysis. In *Int. Joint Conf. on Neural Networks (IJCNN 2009)*, pages 3038–3045, 14–19 June 2009. ISSN: 1098-7576.
- [SL09b] I. Szita and A. Lőrincz. Optimistic initialization and greediness lead to polynomial time learning in factored MDPs. In *ICML 2009*, Montreal, 2009. Omnipress.
- [STL03] I. Szita, B. Takács, and A. Lőrincz. Epsilon-MDPs: Learning in varying environments. *J. of Mach. Learn. Res.*, 3:145–174, 2003.
- [Sun07] R. Sun. The importance of cognitive architectures: An analysis based on CLARION. *J. of Exp. and Theor. Artif. Intell.*, 19:159–193, 2007.
- [Tao06] T. Tao. Szemerédi’s regularity lemma revisited. *Contrib. Discrete Math.*, 1:8–28, 2006.
- [VT05] M. A. O. Vasilescu and D. Terzopoulos. Multilinear independent components analysis. In *Proc. Comp. Vision and Pattern Recog.*, 2005.
- [YBK09] K. Yarrow, P. Brown, and J. W. Krakauer. Inside the brain of an elite athlete. *Nature Reviews Neuroscience*, 10:585–596, 2009.

On Super-Turing Computing Power and Hierarchies of Artificial General Intelligence Systems

Jiří Wiedermann

Institute of Computer Science
Academy of Sciences of the Czech Republic
Pod Vodárenskou věží 2, 182 07 Prague

Abstract

Using the contemporary view of computing exemplified by recent models and results from non-uniform complexity theory we investigate the computational power of artificial general intelligence systems (AGISs). We show that in accordance with the so-called Extended Turing Machine Paradigm such systems can be seen as non-uniform evolving interactive systems whose computational power surpasses that of classical Turing machines. Our results shed light to the question asked by R. Penrose concerning the mathematical capabilities of human mathematicians which seem to go beyond classical computability. We also show that there is an infinite hierarchy of AGISs each of which is capable to solve strictly more problems than its predecessors in the hierarchy.

Characterizing the Computational Properties of AGISs

According to its definition artificial general intelligence is a form of intelligence at the human level and definitely beyond. Implicitly, this statement alone evokes an idea of (partially?) ordered “intelligence levels”, one of which should correspond to human intelligence, with still some levels of “superhuman” intelligence above it. The point in time when the “power” of AGISs will reach and trespass the level of human intelligence has obtained a popular label: the Singularity (cf. Kurzweil, 2005). Nevertheless, it seems that in the AI literature there has not been much explicit attention paid to the formal investigation of the “power” and the “levels of intelligence” (in the sense mentioned above) of AGISs. It is the goal of this short notice to present an approach based on the recent developments in the computational complexity theory answering certain questions related to the computational power of AGISs.

Artificial general intelligence systems must clearly be (i) *interactive* — in order to be able to communicate with their environment, to reflect its changes, to get the feedback, etc.; (ii) *evolutionary* — in order to develop over generations, and (iii) potentially *time-unbounded* — in order to allow for their open-ended development.

Therefore AGISs cannot be modelled by classical Turing machines — simply because such machines do not possess the above mentioned properties. The AGISs must be modelled by theoretical computational models capturing interactivity, evolvability, and time-unbounded operation of the

underlying systems. Such models have recently been introduced by van Leeuwen & Wiedermann, (2001) or (2008).

Definition 1 *An interactive Turing machine with advice is a Turing machine whose architecture is changed in two ways:*

- *instead of an input and output tape it has an input port and an output port allowing reading or writing potentially infinite streams of symbols;*
- *the machine is enhanced by a special, so-called advice tape that, upon a request, allows insertion of a possibly non-computable external information that takes a form of a finite string of symbols. This string must not depend on the concrete stream of symbols read by the machine until that time; it can only depend on the number of those symbols.*

An advice is different from an oracle also considered in the computability theory: an oracle value can depend on the current input (cf. Turing, 1939). The interactive Turing machines with advice represent a *non-uniform model of interactive, evolving, and time-unbounded computation*. Such machines capture well an interactive and time-unbounded software evolution of AGISs.

Interactive Turing machines with advice are equivalent to so-called *evolving automata* that capture well hardware evolution of interactive and time-unbounded computations (Wiedermann & van Leeuwen, 2008).

Definition 2 *The evolving automaton with a schedule is an infinite sequence of finite automata sharing the following property: each automaton in the sequence contains some subset of states of the previous automaton in that sequence. The schedule determines when an automaton has to stop processing of its inputs and thus, when is the turn of the next automaton.*

The condition that a given automaton has among its states a subset of states of a previous automaton captures one important aspect: it is the persistence of data in the evolving automaton over time. In the language of finite automata this condition ensures that some information available to the current automaton is also available to its successor. This models passing of information over generations.

On an on-line delivered potentially infinite sequence of the inputs symbols the schedule of an evolving automaton

determines the *switching times* when the inputs to an automaton must be redirected to the next automaton. This feature models the (hardware) evolution.

An evolving automaton is an infinite object given by an explicit enumeration of all its elements. There may not exist an algorithm enumerating the individual automata. Similarly, the schedule may also be non-computable. Therefore, also evolving automata represent a non-uniform, interactive evolutionary computational model.

Note that at each time a computation of an evolving automaton is performed by exactly one of its elements (one automaton) which is a finite object.

Based on the previous two models van Leeuwen & Wiedermann (2001) have formulated the following thesis:

Extended Turing Machine Paradigm *A computational process is any process whose evolution over time can be captured by evolving automata or, equivalently, by interactive Turing machines with advice.*

Interestingly, the paradigm also expresses the equivalence of software and hardware evolution.

In Wiedermann & van Leeuwen (2008) the authors have shown that the paradigm captures well the contemporary ideas on computing. The fact that it also covers AGISs adds a further support to this paradigm.

Thesis 3 *From a computational point of view AGISs are equivalent to either evolving automata or interactive Turing machines with advice.*

The Super-Turing Computing Power of AGISs

The power of artificial general intelligent systems is measured in terms of sizes of sets of different reactions (or behaviors) that those systems can produce in potentially infinite interactions with their environment.

The super-Turing power of AGISs is shown by referring to super-Turing computing power of interactive Turing machines with advice.

Namely, in van Leeuwen & Wiedermann (2001) it was shown that such machines can solve the halting problem. In order to do so they need an advice that for each input of size n allows to stop their computation once it runs beyond a certain maximum time. This time is defined as the maximum, over computations over all inputs of size n and over all machines of size n that halt on such inputs.

Proposition 4 *The artificial general intelligence systems have super-Turing computational power.*

Roger Penrose (1994) asked about the power of human thoughts: how to explain the fact that mathematicians are able to find proofs of some theorems in spite of the fact that in general (by virtue of Gödel's or Turing's results) there is no algorithm that would always lead to a proof or refutation of any theorem. In our setting the explanation could be that the mathematicians discover a "non-uniform proof", i.e., a way of proving a particular theorem at hand and probably nothing else. This proof is found in a non-predictable potentially unbounded interaction of mathematicians (among themselves and also in the interaction with others and with their environment) pondering over the respective problems.

Hierarchies of AGISs

For interactive Turing machines with advice or for evolving automata one can prove that there exist infinite proper hierarchies of computational problems that can be solved on some level of the hierarchy but not on any of the lower levels. Roughly speaking, the bigger the advice, the more problems can be solved by the underlying machine.

Proposition 5 *There is infinity of infinite proper hierarchies of artificial general intelligence systems of increasing computational power.*

Among the levels of the respective hierarchies there are many levels corresponding formally (and approximately) to the level of human intelligence (the Singularity level) and also infinitely many levels surpassing it in various ways.

Common Pitfalls in Interpretations of the Previous Results

Our results are non-constructive — they merely show the existence of AGISs with super-Turing properties, but not the ways how to construct them. Whether such systems will find solutions of non-computable problems depends on the problem at hand and on getting a proper idea at due time stemming from the sufficient experience, insight and a lucky interaction.

Whether a Singularity will ever be achieved cannot be guaranteed; from our results we merely know that in principle it exists. Our results give no hints how far in the future it lies. Moreover, we have no idea how far apart are the levels in the respective hierarchies. It is quite possible that bridging the gap between the neighboring "interesting" levels of intelligence could require an exponential (or greater) computational effort. Thus, even an exponential development of non-biological intelligence of the AGISs may not help to overcome this gap in a reasonable time.

Acknowledgment

This research was carried out within the institutional research plan AV0Z10300504 and partially supported by a GA ČR grant No. P202/10/1333

References

- Kurzweil, R. (2005). *The Singularity is Near*. Viking Books, 652 pages
- Penrose, R. (1994). *Shadows of the Mind (A Search for the Missing Science of Consciousness)*. Oxford University Press, Oxford, 457 p.
- Turing, A. M. (1939). Systems of logic based on ordinals, *Proc. London Math. Soc. Series 2*, Vol. 45, pp. 161-228
- van Leeuwen, and Wiedermann, J. (2001). The Turing machine paradigm in contemporary computing, *Mathematics unlimited - 2001 and beyond*, Springer-Verlag, pp. 1139-1155
- Wiedermann, J., and van Leeuwen, J. (2008). How We Think of Computing Today. (Invited Talk) *Proc. CiE 2008, LNCS 5028*, Springer, Berlin, pp. 579-593

On Evaluating Agent Performance in a Fixed Period of Time

José Hernández-Orallo

DSIC, Univ. Politècnica de València,
Camí de Vera s/n, 46020 Valencia, Spain. jorallo@dsic.upv.es

Abstract

The evaluation of several agents over a given task in a finite period of time is a very common problem in experimental design, statistics, computer science, economics and, in general, any experimental science. It is also crucial for intelligence evaluation. In reinforcement learning, the task is formalised as an interactive environment with observations, actions and rewards. Typically, the decision that has to be made by the agent is a choice among a set of actions, cycle after cycle. However, in real evaluation scenarios, the time can be *intentionally* modulated by the agent. Consequently, agents not only choose an action but they also choose the time when they want to perform an action. This is natural in biological systems but it is also an issue in control. In this paper we revisit the classical reward aggregating functions which are commonly used in reinforcement learning and related areas, we analyse their problems, and we propose a modification of the average reward to get a consistent measurement for continuous time.

Introduction

Measuring agent intelligence is one of the pending sub-tasks (or requirements) in the goal of constructing general intelligent artefacts. (LH07) presents a formal definition of intelligence as the evaluated performance in a broad range of contexts or environments. However, time is disregarded in their definition. In (HOD09), an implementation of an anytime intelligence test is endeavoured, where time is considered. The introduction of time in the evaluation has much more implications than it might seem at first glance. We do not only face the issue that fast agents score better than slow agents, but we also need to assess other problems: how can we evaluate fast and slow agents in the same setting? How can we deal with intelligent agents that make a shrewd use of response times to score better?

These problems have not been solved in AI areas where agent evaluation is custom. For instance, evaluating decision-making agents in interactive environments where observations, actions and rewards take place has been a well-studied problem in the area of reinforcement learning (SB98). But, in general, time

(either discrete or continuous) is understood as a virtual time. Even in real applications, where continuous time appears, any performance evaluation based on rewards typically does not consider the decision-making time of the agents and, to our knowledge, never considers extreme speed differences between the agents.

In order to illustrate the problem, imagine that a test (composed of several exercises) is passed to several students. All exercises deal about the same (previously unknown) subject, so typically a good student would improve as s/he does more exercises. Each student receives the first exercise, works on it and writes the result and gets an evaluation score or points (e.g. between 0 and 1). Immediately a second exercise is given and the student works on it similarly. The test goes on until a (previously unknown) time limit τ is reached.

Consider a test taken in half an hour, where several students have got different results, as shown in Figure 1. Who is best? We can say that s_1 usually scores better than s_2 does but s_2 is faster. Let us make the question a little bit more difficult. What about a third student s_3 only being able to complete five exercises? From the figure, we can say that it has done all of them right from almost the beginning. We can also say it is very slow, but with only two attempts she or he has been able to find the way to solve the rest of exercises. And now a more incisive question: what about a fourth student s_4 , who does exercises very fast, but at random, and, eventually, in a series of 5,000 exercises done in the half an hour is able to score well on 50 of them?

In the previous example we can either accumulate the results (so students s_1 , s_2 , s_3 and s_4 would get a total return of 10, 18, 4, 50 respectively) or average the results by the number of exercises (so we would get an average return of $\frac{2}{3}$, $\frac{3}{7}$, $\frac{4}{5}$, $\frac{1}{100}$ respectively). We can also consider the physical time (which is equal for all), and average the results by time, getting a scaling of the total returns, i.e., 20, 36, 8, 100 points per hour.

An opinion here would be to say that speed and performance are two different things that we should weight into an equation which matches the context of application. In the previous case, if the average by exercises is v , the number of exercises is n and τ is the total time a possible formula might be $v' = v \times \sqrt{n}/\tau$, giving values

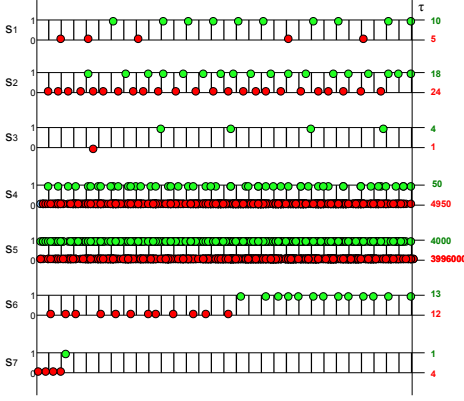


Figure 1: Several students evaluated in a fixed time.

2.3, 2.26, 1.6 and 1 for students s_1 , s_2 , s_3 and s_4 .

The problem is that there is no formula which is valid in general, for different tasks and kinds of agents. Consequently, in this setting, the way in which performance is measured is always task-dependent. But worse, the compensation between v , n and τ is typically non-linear, making different choices when units change, or the measure gives too much weight to speed. Additionally, when $\tau \rightarrow \infty$ the measure goes to 0 (or diverges), against the intuition that the larger the time given the better the evaluation. But the main problem of using time is that for every function which is increasing on speed (n/τ), there is always a very fast agent with a very small average reward, such that it gets better and better scores. Consider, for instance, a student s_5 who does 4,000,000 exercises at random in the half an hour, and is able to score 1 in 4,000 of them and 0 for the rest. The value would be $\frac{1}{1000} \times \frac{2000}{0.5} = 4$. With a very low average performance ($\frac{1}{1000}$), this student gets the best result.

To make things still worse, compare s_3 with s_6 as shown in Figure 1. The speed of s_6 is more than six times greater than s_3 's, but s_3 reaches a state where results are always 1 in about 10 minutes, while s_6 requires about 17 minutes. But if we consider speed, s_6 has a value $v' = \frac{16}{25} \times \frac{5}{0.5} = 5.2$ (while it was 1.6 for s_3).

But in order to realise that this apparently trivial problem is a challenging one, consider another case. Student s_7 acts randomly but she or he modulates time in the following way: whenever the result is 1 then she or he stops doing exercises. If the result is 0 then more exercises are performed very quickly until a 1 is obtained. Note that this strategy scores much better than random in the long term. This means that an opportunistic use of the times could mangle the measurement and convey wrong results.

The previous example tries to informally illustrate the goal and the many problems which arise around agent evaluation in a finite time τ . Simple alternatives such as using fixed time slots are not reasonable, since we want to evaluate agents of virtually any speed, without making them wait. A similar (and simpler) approach is to set a maximum of cycles n instead of a time

τ , but this makes testing almost unfeasible if we do not know the speed of the agent in advance (the test could last milliseconds or years).

As apparently there is no trivial solution, in this paper we want to address the general problem of measuring performance in a time τ under the following setting:

- The overall allotted evaluation time τ is variable and independent of the environment and agent.
- Agents can take a variable time to make an action, which can also be part of their policy.
- The environment must react immediately (no delay time computed on its side).
- The larger the time τ the better the assessment should be (in terms of reliability). This would allow the evaluation to be anytime.
- A constant rate random agent π_{rand}^r should have the same expected value for every τ and rate r .
- The evaluation must be fair, avoiding opportunistic agents, which start with low performance to show an impressive improvement later on, or that stop acting when they get good results (by chance or not).

The main contribution of this work is that we revisit the classical reward aggregation (payoff) functions which are commonly used in reinforcement learning and related areas for our setting (continuous time on the agent, discrete on the environment), we analyse the problems of each of them and we propose a new modification of the average reward to get a consistent measurement for this case, where the agent not only decides an action to perform but also decides the time the decision is going to take.

Setting Definition and Notation

An environment is a world where an agent can interact through actions, rewards and observations. The set of interactions between the agent and the environment is a decision process. Decision processes can be considered discrete or continuous, and stochastic or deterministic.

In our case, the sequence of events is exactly the same as discrete-time decision process. Actions are limited by a finite set of symbols A , (e.g. $\{left, right, up, down\}$), rewards are taken from any subset R of rational numbers, and observations are also limited by a finite set O of possibilities. We will use a_i , r_i and o_i to (respectively) denote action, reward and observation at interaction or cycle (or, more loosely, state) i , with i being a positive natural number. The order of events is always: reward, observation and action. A sequence of k interactions is then a string such as $r_1 o_1 a_1 r_2 o_2 a_2 \dots r_k o_k a_k$. We call these sequence histories, and we will use the notation $\widetilde{roa}_{\leq k}$, $\widetilde{roa}'_{\leq k}$, \dots , to refer to any of these sequences of k interactions and $\widetilde{ro}_{\leq k}$, $\widetilde{ro}'_{\leq k}$, \dots , to refer to any of these sequences just before the action, i.e. $r_1 o_1 a_1 r_2 o_2 a_2 \dots r_k o_k$. Physical time is measured in seconds. We denote by t_i the total physical time elapsed until a_i is performed by the agent.

Both the agent and the environment are defined as a probabilistic measure. In this way, an environment μ is a probabilistic measure which assigns probabilities to each possible pair of observation and reward. For instance, $\mu(r_k o_k | \widetilde{roa}_{\leq k-1})$ denotes the probability in environment μ of outputting $r_k o_k$ after the sequence of events $\widetilde{roa}_{\leq k-1}$. For the agent, though, this is now different to the typical reinforcement learning setting (and more similar to control problems). Given an agent, denoted by π , the term $\pi(d, a_k | \widetilde{ro}_{\leq k})$ denotes the probability of π to execute action a_k before a time delay d after the sequence of events or history $\widetilde{ro}_{\leq k}$. Note that the probability on d is cumulative. Agents can *stop*, i.e., there might be some event sequence $\widetilde{ro}_{\leq k}$ such that $p(d, a_k | \widetilde{ro}_{\leq k}) = 0$ for all d and a_k . Agents, in general, can use information from its previous rewards and observations to determine its future actions and times, i.e. $t_{i+1} - t_i$ can depend on the previous experience.

Interactions between environments and agents can be interrupted at any time τ , known as the “overall or total test time”. The value τ is unknown for any agent at any moment. With $n_\tau^\pi \mu$ (or just n_τ) we denote the number of interactions or cycles performed by π in μ in time τ . Let us see a very simple environment and agent:

Example Consider a test setting where a robot (the agent) can press one of three possible buttons ($A = \{B_1, B_2, B_3\}$), rewards are just a variable score ($R = [0 \dots 1]$) and the observation is two cells where a ball must be inside one of them ($O = \{C_1, C_2\}$). Given the sequence of events so far is $r_1 o_1 a_1 r_2 o_2 a_2 \dots r_{k-1} o_{k-1} a_{k-1}$, we define the environment behaviour as follows:

- If $(a_{k-1} = B_1 \text{ and } o_{k-1} = C_1)$ or $(a_{k-1} = B_2 \text{ and } o_{k-1} = C_2)$ then we generate a raw reward of +0.1.
- Otherwise the raw reward is 0.

The observation o_k in both cases above is generated with the following simple rule: if k is even then $o_k = C_2$. Otherwise, $o_k = C_1$. The first reward (r_1) is 0.

From the previous example, a robot π_1 always pressing button B_1 at a rate of three times per second would have the following interaction: $0C_1B_10.1C_2B_10C_1B_10.1 \dots$ with times $t_i = \frac{1}{3}i$. A second robot π_{rand} presses buttons at random among $\{B_1, B_2, B_3\}$ at a rate $t_i = \frac{1}{10}i$.

Payoff and Environment Classes

Let us give the simplest notion of payoff:

Definition The total reward sum of agent π in environment μ in a fixed time τ is defined as follows¹:

$$V_\mu^\pi \uparrow \tau := E \left(\sum_{i=1}^{n_\tau} r_i \right)$$

¹ $E(\cdot)$ denotes the expected value, which is only necessary in the definition when either (or both) the agent or the environment are non-deterministic.

For the previous example, the total reward for π_1 in 30 seconds would be $\frac{1}{2} \times 30 \times 3 \times 0.1 + \frac{1}{2} \times 30 \times 3 \times 0 = 4.5$. The total reward for π_{rand} in 30 seconds would be $\frac{1}{3} \times 30 \times 10 \times 0.1 + \frac{2}{3} \times 30 \times 10 \times 0 = 10$.

One of the problems of a cumulative reward function is that the greater the time τ the greater the expected value. More precisely, this is always the case only when rewards are positive. Consequently, the previous measure cannot be used as a value in an anytime test where the larger the time τ the better the assessment.

One attempt to solve this problem without abandoning the idea of summing rewards is the notion of reward-bounded (or summable) environment (LH07).

Definition An environment μ is reward-bounded if $\forall i : 0 \leq r_i \leq 1$ and for every agent π :

$$\lim_{\tau \rightarrow \infty} V_\mu^\pi \uparrow \tau = \sum_{i=1}^{\infty} r_i \leq 1$$

The idea is motivated by the issue that payoff functions based on weighted or discounted rewards usually require the arbitrary choice of a discounting function and a parameter. However, the previous idea has several problems. First, it is clear that it is easy to make any environment reward-bounded, by just dividing raw rewards by expressions such as 2^i or any other kind of discounting function whose total sum is lower than 1 (see (Hut06) for an extensive list of possible discounting functions). But this implies that the discount function is hardwired in the environment. We can make this depend on a universal distribution over the universal machine which generates the environments, but in the end this is basically the same as not setting the reward-bounded condition and choose the discount function *externally* with a universal distribution over a universal machine generating discount functions.

In any case, be it internally hardwired in the environment or chosen externally there is another problem with discount functions. For the overwhelming majority of reward-bounded environments, the first actions are typically astronomically more important than the rest. This can be softened with discount functions that approach a uniform distribution or that depend on the agent’s age, but in the end, as the number of interactions grow, the first actions (dozens or millions) get most of the distribution and hence most of the total reward. And, typically the first actions take place when the agent explores the environment. This is related to a similar problem for discounted rewards².

There is still another (more serious) problem. With reward-bounded environments, random agents typically increase their return as τ grows (this also happens for non-random agents, but this is somehow expected). This is against the natural constraint that a constant-rate random agent π_{rand}^r should have the same expected

²In fact, in order to show the equivalence in the limit of the average reward and the discounted reward, (Hut06) infinite many cycles have to be removed from the start.

valued for every τ and this value should also be the same for every rate r .

And, finally, consider the previous aggregated function applied to biological systems (e.g. a child or a chimpanzee). Since all the rewards are always positive, the subject will strive to accumulate as much reward as possible, generally acting fast but rashly (hyperactive).

As an alternative to discounting and also to reward-bounded environments, and especially conceived to work well with any agent, in (HOD09) we propose this:

Definition An environment μ is balanced if $\forall i : -1 \leq r_i \leq 1$ and for a constant-rate random agent π_{rand}^r at any rate r then

$$\forall \tau > 0 : E \left(V_{\mu}^{\pi_{rand}^r} \uparrow \tau \right) = E \left(\sum_{i=1}^{\lfloor r \times \tau \rfloor} r_i \right) = 0$$

The construction of balanced environments is not difficult, even universal ones, as shown in (HO09a). It is clear to see that changing rewards from the interval $[0, 1]$ to the interval $[-1, 1]$ creates a phenomenon which is frequently ignored in reinforcement learning but is omnipresent in economics: “everything that has been earned in previous cycles can be lost afterwards”.

As mentioned in the introduction, the goal was to measure the performance of an agent in an environment in a given time τ . Apart from the unweighted sum, there are many different ways to compute the payoff (or aggregated reward, or return value) of a set of interactions against an environment. In reinforcement learning there are two main approaches for doing that: the cumulative reward (with weights, typically known as discounting) and the average reward (Mah96)(Ber95)(KLM96)(SB98).

Let us see some of them adapted to our continuous time limit setting. For instance, reward can be averaged in two different ways, by the number of cycles of the agent (average reward per cycle), or by the physical elapsed time (average reward per second). Since the second boils down to $\frac{1}{\tau} V_{\mu}^{\pi} \uparrow \tau$ (so inheriting most of the problems of $V_{\mu}^{\pi} \uparrow \tau$), we will just analyse the first.

Definition The average reward per cycle of agent π in environment μ in a fixed time τ is defined as follows:

$$v_{\mu}^{\pi} || \tau := E \left(\frac{1}{n_{\tau}} \sum_{i=1}^{n_{\tau}} r_i \right)$$

If $n_{\tau} = 0$, then $v_{\mu}^{\pi} || \tau$ is defined to be 0.

Let us also revisit the most popular aggregated measure in reinforcement learning, known as discounted reward, which is just a weighted sum. We will see a generalised version of discounted reward, following (Hut06). Accordingly, we define $\gamma = (\gamma_1, \gamma_2, \dots)$ with γ_k being positive real numbers (typically with $\gamma_i > \gamma_{i+1}$), as a summable discount sequence in the sense that $\Gamma_k^n := \sum_{i=k}^n \gamma_i < \infty$. If $k = 1$ we simply use Γ^n .

Hence, the discounted reward (per cycle) is:

Definition The discounted reward of agent π in environment μ in a fixed time τ is defined as follows:

$$V_{\mu}^{\pi} | \gamma | \tau := E \left(\frac{1}{\Gamma^{n_{\tau}}} \sum_{i=1}^{n_{\tau}} \gamma_i r_i \right)$$

A typical choice for γ is the geometric discounting ($\gamma_k = \lambda^k, 0 \leq \lambda < 1$). For a more exhaustive list see (Hut06). As the very name says, all of them are discounting, so the first rewards contribute to the aggregated value much stronger than the rest. How much? That depends on the choice of γ . In any case, the result very dependent on the rate. For instance, agents increase their values with increasing values of τ if the environment is not balanced. And even a slightly better than random agent can have better results (although not very good) than a slower but competent agent. An alternative is to define γ as a function of t_i , but in general this has the same behaviour but additionally this creates other problems (stopping policy problems, as we will see in the following section).

The Problem of Time Modulation

The time taken by each agent to perform each action is not necessarily constant. It might depend on the cost of the computation. But, more importantly, it can be *intentionally* modulated by the agent. Thus, agents not only choose an action but they also choose the time they want to devote to an action. This is natural in biological systems but it is also an issue in control. More generally, an agent could decide to stop, which implies stopping any further exploration but also any further reward.

First, we see the notion of “time modulation policy”:

Definition A reasonable time modulation policy for agent π in environment μ evaluated in a fixed time τ is any intentional (or not) assignment for values t_1, t_2, \dots where $\forall i t_i > t_{i-1}$, such that every t_i can depend on previous t_1, t_2, \dots, t_{i-1} and also on previous rewards and observations, but never on τ (since τ is not known).

A time modulation policy can make the agent stop on t_i (and, hence t_{i+1} is infinite). In our setting, a tricky (but good) policy here would be to act as a fast random agent until having an average reward over a threshold and then stop acting. We call this agent an opportunistic fast random agent. If the threshold is 0 this strategy ensures a positive reward in balanced environments³. Consequently, an agent could get a very good result by having very fast (and possibly lucky) first interactions and then rest on its laurels, because the average so far was good. The following theorem formalises this:

Theorem 1⁴ *There are random agents π_{rand} using stopping policies not knowing τ such that for some balanced environment μ , there is a value t such that $\forall \tau \geq t : v_{\mu}^{\pi_{rand}} || \tau > 0$.*

³In fact, if only rewards -1 and 1 are possible, the expected reward is $0.79 \times 2 - 1 = 0.58$ (see (Fer04)).

⁴Due to space limitations, proofs are found in (HO09b).

A first (and naïve) idea to avoid stopping policies would be to give less weight to quick actions and more weight to slow actions. Apart from being counterintuitive, this would also be tricky, because an agent which is sure of a good action will delay the action as much as possible, which is, again, counterintuitive. On the other hand, giving more weight to quick decisions is more intuitive, but very fast mediocre agents can score well, and, additionally, it also suffers the problems of opportunistic time modulation. A better possibility is shown next:

Definition The average reward per cycle with diminishing history of agent π in environment μ in a fixed time τ is defined as follows:

$$\check{v}_\mu^\pi || \tau := E \left(\frac{1}{n^*} \sum_{i=1}^{n^*} r_i \right) \text{ where } n^* = \left\lfloor n_\tau \left(\frac{t_{n_\tau}}{\tau} \right) \right\rfloor$$

This definition reduces the number of evaluated cycles proportionally to the elapsed time from the last action until τ . If the last actions have been good and we delay future actions and let time pass, we soon make the measure ignore these recent good rewards. If we stop, in the limit, the measure reaches 0, so it also avoids stopping policies, as the following theorem shows.

Theorem 2 *For every balanced environment μ and every agent π , there is no stopping policy not knowing τ which eventually stops such that π_{rand} has $\lim_{\tau \rightarrow \infty} \check{v}_\mu^{\pi_{rand}} || \tau > 0$.*

And now, we can ensure what happens in any case (stopping or not) for a constant-rate random agent:

Theorem 3 *For every balanced environment μ , a constant-rate random agent π_{rand} with any stopping policy has $\lim_{\tau \rightarrow \infty} \check{v}_\mu^{\pi_{rand}} || \tau = 0$.*

A more difficult question is whether time modulation policies are completely avoided by the previous definition. The answer is no, as we see next.

Lemma 4 *We denote $R_\mu^{\pi_{rand}}(i)$ the result of any given payoff function R until action i . For every R , an agent π after action a_i with a locally optimal time modulation policy should wait a time t_d for the next action if and only if $\forall t_i \leq t < t_i + t_d : R_\mu^{\pi_{rand}}(i) > E(R_\mu^{\pi_{rand}}(i+1))$.*

In other words, the payoff until $t_i + t_d$ not performing any action is greater than the expected payoff performing the following action. The previous lemma does not say whether the agent can know the expected payoff. In fact, even in cases where the overall expected payoff is clear, an agent can use a wrong information and make a bad policy. Note that lemma 4 is shown with the true expected value, and not the expected or estimated value by the agent. With this, we can conclude that although random agents can use time modulation policies and can work well in some environments, they can also be bad in other environments. As a result, good agents can also be discriminated from bad agents because they have (or not) good modulation policies. The following theorem shows that good time modulation policies are not easy to find, in general.

Theorem 5 *Given any agent π there is no time modulation policy which is optimal for every balanced environment μ .*

So we have realised that time modulations are impossible to avoid (only minimise). As a result, we will have to accept time modulation as part of the agent behaviour and needs to be considered in the measurement.

Comparison of Payoff Functions

After the analysis of several payoff functions adapted from the literature, and the introduction of a new variant with some associated results, it is necessary to recapitulate and give a comprehensive view. The setting we introduce in this paper is characterised by different response times on the side of the agent. These different response times could be motivated by different agent speeds or by an intentional use of delays.

Other practical issues for each function are related to the behaviour against random agents, the convergence or boundedness of the results, whether there is a preference for the start of the testing period, etc. In what follows, we will examine the previous payoff functions according to several features, as shown in table 1.

There are several measures which cluster together. For instance $V_\mu^\pi \uparrow \tau$ and $\omega_\mu^\pi \tau$ get almost the same answers, since one is the scaling of the other using τ . And $V_\mu^\pi \uparrow \tau$ also gets very similar results to $V_\mu^\pi | \gamma | \tau$, since all of them are cumulative. Averages, on the contrary, have a different pattern. In general, it is also remarkable that the use of balanced environments typically is more problematic on issues 10 and 11, while being better on 1 and 2. The measure \check{v} in balanced environments gets 11 ‘yes’ from a total of 12.

Feature 9 has to be discussed in more detail. It refers to cases where necessarily (not because of the agent’s time modulation policy) the response times increase with time. This is a general issue in many problems, since, as time increases, more history has to be taken into account and decisions can be more difficult to make. Consider for instance a problem such that choosing the right decision at interaction i has an increasing polynomial time complexity. Consequently, many of the payoff functions will penalise the agent executing this algorithm for increasing values of τ or n_τ . On the contrary, $v_\mu^\pi || \tau$ would not penalise this at all (but allows the stopping problem) and $\check{v}_\mu^\pi || \tau$ penalises it very mildly. For problems with exponential complexity (and many other NP-problems), though, $\check{v}_\mu^\pi || \tau$ typically will make n^* go to zero between interactions ($t_{i+1} > 2t_i$). This means that other algorithms approximating the problem in polynomial time could get better rewards.

Conclusions

This paper has addressed a problem which is apparently trivial: to evaluate the performance of an agent in a finite period of time, considering that agent actions can take a variable time delay (intentionally or not). However, the evaluation is more cumbersome than it

Environment Type	General	Bounded	Balanced	General	Balanced	General	Balanced	Balanced
Score Function	$V_\mu^\pi \uparrow \tau$	$V_\mu^\pi \uparrow \tau$	$V_\mu^\pi \uparrow \tau$	$v_\mu^\pi \tau$	$v_\mu^\pi \tau$	$V_\mu^\pi \gamma \tau$	$V_\mu^\pi \gamma \tau$	$\tilde{v}_\mu^\pi \tau$
1. Do random agents get a somehow central value (preferably 0)?	No	No	Yes	No	Yes	No	Yes	Yes
2. Is the result of random agents independent from τ and the rate?	No	No	Yes	No	Yes	No	Yes	Yes
3. Is it avoided that a fast mediocre agent can score well?	No	No	No	Yes	Yes	No	No	Yes
4. Does the measurement work well when rates $\rightarrow \infty$?	No	No	No	Yes	Yes	No	No	Yes
5. Do better but slower agents score better than worse but faster agents?	No	No	No	Yes	Yes	*	*	Yes
6. Do faster agents score better than slow ones with equal performance?	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
7. Are the first interactions as relevant as the rest?	Yes	No	Yes	Yes	Yes	No	No	Yes
8. Is the measure bounded for all τ ?	No	Yes	No	Yes	Yes	Yes	Yes	Yes
9. Does it work well when actions require more and more time to decide?	No	No	No	Yes	Yes	No	No	Yes
10. Is it robust against time stopping policies?	Yes	Yes	No	No	No	Yes	No	Yes
11. Is it robust against time modulation policies?	Yes	Yes	No	No	No	Yes	No	No
12. Is it scale independent (different time units)?	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes

Table 1: Comparison of Payoff Functions. Symbol ‘*’ denotes that it may depend on a parameter (e.g. γ).

might seem at first sight. First of all, it is closely related but not the same as the measurement in reinforcement learning, which typically disregards agent reaction times. Additionally, payoff functions are conceived to be embedded in the design of the algorithms that control agent behaviour, not to be used in a general test setting. And it is important to mention this again, since here we are not (mainly) concerned with the design of agents but in their evaluation. Consequently, we know that, as Hutter says (Hut06): “eternal agents are lazy”, and might procrastinate their actions. This is what typically happens with averages, since with an infinite number of cycles (i.e. eternal life) we will always be able to compensate any initial bad behaviour. We do not want to avoid this. We want that, if this happens, the measure takes it into account. When the lifetime τ is not known or is infinite, a typical possibility is to use a weighting (i.e. discounting). This generally translates into an evaluation weighting where the first actions are more important than the rest, which is not reasonable. This does not mean that the formula of discounted reward should not be used in agent design. On the contrary, discounted reward and the techniques that derive from them (such as Q-learning) could work well in our setting, but we should not use them as the external performance measure. In any case, we must devise tests that work with artificial agents but also with biological beings. This is one of the reasons that negative rewards are needed. Paraphrasing Hutter, we can say that “using cumulative positive rewards make agents hyperactive”.

Our main concern, however, has been an opportunistic use of time. This problem does not exist when using discrete-time agents and it is uncommon in evaluation, especially outside control and robotics, where the goals and measurements are different. The adjustment proposal on the average tries to solve the stopping problem.

The main application of our proposal is for measuring performance in a broad range of environments which, according to (LH07), boils down to measuring intelligence. The setting which is presented here is necessary for an anytime intelligence test (HOD09), where the evaluation can be stopped anytime, and the results should be better the more time we have for the test.

Finally, as future work, the use of continuous-time en-

vironments must be investigated, especially when other agents can play inside the environment. This is typical in multi-agent systems. The problem here is to determine the rate of the system, because it can be too fast for some agents and too slow for others.

Acknowledgments

The author thanks the funding from the Spanish Ministerio de Educación y Ciencia (MEC) for projects Explora-Ingenio TIN2009-06078-E, Consolider 26706 and TIN 2007-68093-C02, and Prometeo/2008/051 from GVA.

References

- D.P. Bertsekas. *Dynamic Programming and Optimal Control*. Athena Scientific, Belmont, MA, 1995.
- T. S. Ferguson. *Optimal Stopping and Applications*. Maths. Department, UCLA, 2004. <http://www.math.ucla.edu/~tom/Stopping/Contents.html>.
- J. Hernández-Orallo. A (hopefully) unbiased universal environment class for measuring intelligence of biological and artificial systems. Extended Version. *available at* <http://users.dsic.upv.es/proy/anynt/>, 2009.
- J. Hernández-Orallo. On evaluating agent performance in a fixed period of time. Ext.d. Version. *available at* <http://users.dsic.upv.es/proy/anynt/>, 2009.
- J. Hernández-Orallo and D. L. Dowe. Measuring universal intelligence: Towards an anytime intelligence test. *Under Review, available at* <http://users.dsic.upv.es/proy/anynt/>, 2009.
- M. Hutter. General discounting versus average reward. In *ALT*, volume 4264 of *LNCS*, pages 244–258. Springer, 2006.
- L.P. Kaelbling, M.L. Littman, and A.W. Moore. Reinforcement learning: A survey. *J. Artif. Intel.*, 4(1):237–285, 1996.
- S. Legg and M. Hutter. Universal intelligence: A definition of machine intelligence. *Minds & Mach.*, 17(4):391–444, 2007.
- S. Mahadevan. Average reward reinforcement learning: Foundations, algorithms, empirical results. *Mach. Learn.*, 22, 1996.
- R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, March 1998.

A conversion between utility and information

Pedro A. Ortega

Department of Engineering
University of Cambridge
Cambridge CB2 1PZ, UK
peortega@dcc.uchile.cl

Daniel A. Braun

Department of Engineering
University of Cambridge
Cambridge CB2 1PZ, UK
dab54@cam.ac.uk

Abstract

Rewards typically express desirabilities or preferences over a set of alternatives. Here we propose that rewards can be defined for any probability distribution based on three desiderata, namely that rewards should be real-valued, additive and order-preserving, where the latter implies that more probable events should also be more desirable. Our main result states that rewards are then uniquely determined by the negative information content. To analyze stochastic processes, we define the utility of a realization as its reward rate. Under this interpretation, we show that the expected utility of a stochastic process is its negative entropy rate. Furthermore, we apply our results to analyze agent-environment interactions. We show that the expected utility that will actually be achieved by the agent is given by the negative cross-entropy from the input-output (I/O) distribution of the coupled interaction system and the agent's I/O distribution. Thus, our results allow for an information-theoretic interpretation of the notion of utility and the characterization of agent-environment interactions in terms of entropy dynamics.

Keywords: Behavior, Utility, Entropy, Information

Introduction

Purposeful behavior typically occurs when an agent exhibits specific preferences over different states of the environment. Mathematically, these preferences can be formalized by the concept of a utility function that assigns a numerical value to each possible state such that states with higher utility correspond to states that are more desirable [Fishburn, 1982]. Behavior can then be understood as the attempt to increase one's utility. Accordingly, utility functions can be measured experimentally by observing an agent choosing between different options, as this way its preferences are revealed. Mathematical models of rational agency that are based on the notion of utility have been widely applied in behavioral economics, biology and artificial intelligence research [Russell and Norvig, 1995]. Typically, such rational agent models assume a distinct reward signal (or cost) that an agent is explicitly trying to optimize.

However, as an observer we might even attribute purposefulness to a system that does not have an explicit

reward signal, because the dynamics of the system itself reveal a preference structure, namely the preference over all possible paths through history. Since in most systems not all of the histories are equally likely, we might say that some histories are more probable than others because they are more desirable from the point of view of the system. Similarly, if we regard all possible interactions between a system and its environment, the behavior of the system can be conceived as a drive to generate desirable histories. This imposes a conceptual link between the probability of a history happening and the desirability of that history. In terms of agent design, the intuitive rationale is that agents should act in a way such that more desired histories are more probable. The same holds of course for the environment. Consequently, a competition arises between the agent and the environment, where both participants try to drive the dynamics of their interactions to their respective desired histories. In the following we want to show that this competition can be quantitatively assessed based on the entropy dynamics that govern the interactions between agent and environment.

Preliminaries

We introduce the following notation. A set is denoted by a calligraphic letter like \mathcal{X} and consists of *elements* or *symbols*. *Strings* are finite concatenations of symbols and *sequences* are infinite concatenations. The empty string is denoted by ϵ . \mathcal{X}^n denotes the set of strings of length n based on \mathcal{X} , and $\mathcal{X}^* \equiv \bigcup_{n \geq 0} \mathcal{X}^n$ is the set of finite strings. Furthermore, $\mathcal{X}^\infty \equiv \{x_1 x_2 \dots | x_i \in \mathcal{X} \text{ for all } i = 1, 2, \dots\}$ is defined as the set of one-way infinite sequences based on \mathcal{X} . For substrings, the following shorthand notation is used: a string that runs from index i to k is written as $x_{i:k} \equiv x_i x_{i+1} \dots x_{k-1} x_k$. Similarly, $x_{\leq i} \equiv x_1 x_2 \dots x_i$ is a string starting from the first index. By convention, $x_{i:j} \equiv \epsilon$ if $i > j$. All proofs can be found in the appendix.

Rewards

In order to derive utility functions for stochastic processes over finite alphabets, we construct a utility function from an auxiliary function that measures the de-

desirability of events, i.e. such that we can assign desirability values to every finite interval in a realization of the process. We call this auxiliary function the reward function. We impose three desiderata on a reward function. First, we want rewards to be mappings from events to real numbers that indicate the degree of desirability of the events. Second, the reward of a joint event should be obtained by summing up the reward of the sub-events. For example, the “reward of drinking coffee and eating a croissant” should equal “the reward of drinking coffee” plus the “reward of having a croissant given the reward of drinking coffee”¹. This is the additivity requirement of the reward function. The last requirement that we impose for the reward function should capture the intuition suggested in the introduction, namely that more desirable events should also be more probable events given the expectations of the system. This is the consistency requirement.

We start out from a *probability space* $(\Omega, \mathcal{F}, \mathbf{Pr})$, where Ω is the sample space, \mathcal{F} is a dense σ -algebra and \mathbf{Pr} is a probability measure over \mathcal{F} . In this section, we use lowercase letters like x, y, z to denote the elements of the σ -algebra \mathcal{F} . Given a set \mathcal{X} , its complement is denoted by complement \mathcal{X}^c and its powerset by $\mathcal{P}(\mathcal{X})$. The three desiderata can then be summarized as follows:

Definition 1 (Reward). Let $S = (\Omega, \mathcal{F}, \mathbf{Pr})$ be a probability space. A function \mathbf{r} is a *reward function* for S iff it has the following three properties:

1. *Real-valued*: for all $x, y \in \mathcal{F}$,

$$\mathbf{r}(x|y) \in \mathbb{R};$$

2. *Additivity*: for all $x, y, z \in \mathcal{F}$,

$$\mathbf{r}(x, y|z) = \mathbf{r}(x|z) + \mathbf{r}(y|x, z);$$

3. *Consistent*: for all $x, y, u, v \in \mathcal{F}$,

$$\mathbf{Pr}(x|u) > \mathbf{Pr}(y|v) \iff \mathbf{r}(x|u) > \mathbf{r}(y|v).$$

Furthermore, the unconditional reward is defined as $\mathbf{r}(x) \equiv \mathbf{r}(x|\Omega)$ for all $x \in \mathcal{F}$.

The following theorem shows that these three desiderata enforce a strict mapping rewards and probabilities. The only function that can express such a relationship is the logarithm.

Theorem 1. Let $S = (\Omega, \mathcal{F}, \mathbf{Pr})$ be a probability space. Then, a function \mathbf{r} is a reward function for S iff for all $x, y \in \mathcal{F}$

$$\mathbf{r}(x|y) = k \log \mathbf{Pr}(x|y),$$

where $k > 0$ is an arbitrary constant.

Notice that the constant k in the expression $\mathbf{r}(x|y) = k \log \mathbf{Pr}(x|y)$ merely determines the units in which we choose to measure rewards. Thus, the reward function

¹Note that the additivity property does not imply that the reward for two coffees is simply twice the reward for one coffee, as the reward for the second coffee will be conditioned on having had a first coffee already.

\mathbf{r} for a probability space $(\Omega, \mathcal{F}, \mathbf{Pr})$ is essentially unique. As a convention, we will assume natural logarithms and set the constant to $k = 1$, i.e. $\mathbf{r}(x|y) = \ln \mathbf{Pr}(x|y)$.

This result establishes a connection to information theory. It is immediately clear that the reward of an event is nothing more than its negative information content: the quantity $h(x) = -\mathbf{r}(x)$ is the Shannon information content of $x \in \mathcal{F}$ measured in *nats* [MacKay, 2003]. This means that we can interpret rewards as “negative surprise values”, and that “surprise values” constitute losses.

Proposition 1. Let \mathbf{r} be a reward function over a probability space $(\Omega, \mathcal{F}, \mathbf{Pr})$. Then, it has the following properties:

i. Let $x, y \in \mathcal{F}$. Then

$$-\infty = \mathbf{r}(\emptyset) \leq \mathbf{r}(x|y) \leq \mathbf{r}(\Omega) = 0.$$

ii. Let $x \in \mathcal{F}$ be an event. Then,

$$e^{\mathbf{r}(x^c)} = 1 - e^{\mathbf{r}(x)}.$$

iii. Let $z_1, z_2, \dots \in \mathcal{F}$ be a sequence of disjoint events with rewards $\mathbf{r}(z_1), \mathbf{r}(z_2), \dots$ and let $x = \bigcup_i z_i$. Then

$$e^{\mathbf{r}(x)} = \sum_i e^{\mathbf{r}(z_i)}.$$

The proof of this proposition is trivial and left to the reader. The first part sets the bounds for the values of rewards, and the two latter explain how to construct the rewards of events from known rewards using complement and countable union of disjoint events.

At a first glance, the fact that rewards take on complicated non-positive values might seem unnatural, as in many applications one would like to use numerical values drawn from arbitrary real intervals. Fortunately, given numerical values representing the desirabilities of events, there is always an affine transformation that converts them into rewards.

Theorem 2. Let Ω be a countable set, and let $d : \Omega \rightarrow (-\infty, a]$ be a mapping. Then, for every $\alpha > 0$, there is a probability space $(\Omega, \mathcal{P}(\Omega), \mathbf{Pr})$ with reward function \mathbf{r} such that:

1. for all $\omega \in \Omega$,

$$\mathbf{r}(\{\omega\}) \equiv \alpha d(\omega) + \beta,$$

$$\text{where } \beta \equiv -\ln \left(\sum_{\omega' \in \Omega} e^{\alpha d(\omega')} \right);$$

2. and for all $\omega, \omega' \in \Omega$,

$$d(\omega) > d(\omega') \iff \mathbf{r}(\{\omega\}) > \mathbf{r}(\{\omega'\}).$$

Note that Theorem 2 implies that the probability $\mathbf{Pr}(x)$ of any event x in the σ -algebra $\mathcal{P}(\Omega)$ generated by Ω is given by

$$\mathbf{Pr}(x) = \frac{\sum_{\omega \in x} e^{\alpha d(\omega)}}{\sum_{\omega \in \Omega} e^{\alpha d(\omega)}}.$$

Note that for singletons $\{\omega\}$, $\mathbf{Pr}(\{\omega\})$ is the Gibbs measure with negative energy $d(\omega)$ and temperature $\propto \frac{1}{\alpha}$. It is due to this analogy that we call the quantity $\frac{1}{\alpha} > 0$ the *temperature parameter* of the transformation.

Utilities in Stochastic Processes

In this section, we consider a *stochastic process* \mathbf{Pr} over sequences $x_1x_2x_3\cdots$ in \mathcal{X}^∞ . We specify the process by assigning conditional probabilities $\mathbf{Pr}(x_t|x_{<t})$ to all finite strings $x_{\leq t} \in \mathcal{X}^*$. Note that the distribution $\mathbf{Pr}(x_{\leq t}) = \prod_{\tau=1}^t \mathbf{Pr}(x_\tau|x_{<\tau})$ for all $x_{\leq t} \in \mathcal{X}^*$ is normalized by construction. By the Kolmogorov extension theorem, it is guaranteed that there exists a unique probability space $S = (\mathcal{X}^\infty, \mathcal{F}, \mathbf{Pr})$. We therefore omit the reference to S and talk about the process \mathbf{Pr} .

The reward function \mathbf{r} derived in the previous section correctly expresses preference relations amongst different outcomes. However, in the context of random sequences, it has the downside that the reward of most sequences diverges. A sequence $x_1x_2x_3\cdots$ can be interpreted as a progressive refinement of a point event in \mathcal{F} , namely, the sequence of events $\epsilon \supset x_{\leq 1} \supset x_{\leq 2} \supset x_{\leq 3} \supset \cdots$. One can exploit the interpretation of the index as time to define a quantity that does not diverge. We define thus the utility as the reward rate of a sequence.

Definition 2 (Utility). Let \mathbf{r} be a reward function for the process \mathbf{Pr} . The utility of a string $x_{\leq t} \in \mathcal{X}^*$ is defined as

$$\mathbf{U}(x_{\leq t}) \equiv \frac{1}{t} \sum_{\tau=1}^t \mathbf{r}(x_\tau|x_{<\tau}),$$

and for a sequence $x = x_1x_2x_3\cdots \in \mathcal{X}^\infty$ it is defined as

$$\mathbf{U}(x) \equiv \lim_{t \rightarrow \infty} \mathbf{U}(x_{\leq t})$$

if this limit exists².

A utility function that is constructed according to Definition 2 has the following properties.

Proposition 2. Let \mathbf{U} be a utility function for a process \mathbf{Pr} . The following properties hold:

- i. For all $x = x_1x_2\cdots \in \mathcal{X}^\infty$ and all $t, k \in \mathbb{N}$,
 $-\infty = \mathbf{U}(\lambda) \leq \mathbf{U}(x_{\leq t}) \leq \mathbf{U}(\epsilon) = 0$,

where λ is any impossible string/sequence.

- ii. For all $x_{\leq t} \in \mathcal{X}^*$,

$$\mathbf{Pr}(x_{\leq t}) = \exp(t \cdot \mathbf{U}(x_{\leq t})).$$

- iii. For any $t \in \mathbb{N}$,

$$\mathbf{E}[\mathbf{U}(x_{\leq t})] = -\frac{1}{t} \mathbf{H}[\mathbf{Pr}(x_{\leq t})],$$

where \mathbf{H} is the entropy functional (see the appendix).

Part (i) provides trivial bounds on the utilities that directly carry over from the bounds on rewards. Part (ii) shows how the utility of a sequence determines its probability. Part (iii) implies that the expected utility of an interaction sequence is just its negative entropy rate.

²Strictly speaking, one could define the upper and lower rate $\mathbf{U}^+(x) \equiv \limsup_{t \rightarrow \infty} \mathbf{U}(x_{\leq t})$ and $\mathbf{U}^-(x) \equiv \liminf_{t \rightarrow \infty} \mathbf{U}(x_{\leq t})$ respectively, but we avoid this distinction for simplicity.

Utility in Coupled I/O systems

Let \mathcal{O} and \mathcal{A} be two finite sets, the first being the *set of observations* and the second being the *set of actions*. Using \mathcal{A} and \mathcal{O} , a set of interaction sequences is constructed. Define the *set of interactions* as $\mathcal{Z} \equiv \mathcal{A} \times \mathcal{O}$. A pair $(a, o) \in \mathcal{Z}$ is called an *interaction*. We underline symbols to glue them together as in $\underline{ao}_{\leq t} = a_1o_1a_2o_2\cdots a_t o_t$.

An *I/O system* \mathbf{Pr} is a probability distribution over interaction sequences \mathcal{Z}^∞ . \mathbf{Pr} is uniquely determined by the conditional probabilities

$$\mathbf{Pr}(a_t|\underline{ao}_{<t}), \quad \mathbf{Pr}(o_t|\underline{ao}_{<t}a_t)$$

for each $\underline{ao}_{<t} \in \mathcal{Z}^*$. However, the semantics of the probability distribution \mathbf{Pr} are only fully defined once it is coupled to another system. Note that an I/O system is formally equivalent to a stochastic process; hence one can construct a reward function \mathbf{r} for \mathbf{Pr} .

Let \mathbf{P}, \mathbf{Q} be two I/O systems. An *interaction system* (\mathbf{P}, \mathbf{Q}) defines a *generative distribution* \mathbf{G} that describes the probabilities that actually govern the I/O stream once the two systems are coupled. \mathbf{G} is specified by the equations

$$\begin{aligned} \mathbf{G}(a_t|\underline{ao}_{<t}) &= \mathbf{P}(a_t|\underline{ao}_{<t}) \\ \mathbf{G}(o_t|\underline{ao}_{<t}a_t) &= \mathbf{Q}(o_t|\underline{ao}_{<t}a_t) \end{aligned}$$

valid for all $\underline{ao}_t \in \mathcal{Z}^*$. Here, \mathbf{G} is a stochastic process over \mathcal{Z}^∞ that models the true probability distribution over interaction sequences that arises by coupling two systems through their I/O streams. More specifically, for the system \mathbf{P} , $\mathbf{P}(a_t|\underline{ao}_{<t})$ is the probability of producing action $a_t \in \mathcal{A}$ given history $\underline{ao}_{<t}$ and $\mathbf{P}(o_t|\underline{ao}_{<t}a_t)$ is the predicted probability of the observation $o_t \in \mathcal{O}$ given history $\underline{ao}_{<t}a_t$. Hence, for \mathbf{P} , the sequence $o_1o_2\cdots$ is its input stream and the sequence $a_1a_2\cdots$ is its output stream. In contrast, the roles of actions and observations are reversed in the case of the system \mathbf{Q} . This model of interaction is very general in that it can accommodate many specific regimes of interaction. By convention, we call the system \mathbf{P} the *agent* and the system \mathbf{Q} the *environment*.

In the following we are interested in understanding the actual utilities that can be achieved by an agent \mathbf{P} once coupled to a particular environment \mathbf{Q} . Accordingly, we will compute expectations over functions of interaction sequences with respect to \mathbf{G} , since the generative distribution \mathbf{G} describes the actual interaction statistics of the two coupled I/O systems.

Theorem 3. Let (\mathbf{P}, \mathbf{Q}) be an interaction system. The expected rewards of \mathbf{G} , \mathbf{P} and \mathbf{Q} for the first t interactions are given by

$$\begin{aligned} \mathbf{E}[\mathbf{r}_\mathbf{G}(\underline{ao}_{\leq t})] &= -\mathbf{H}[\mathbf{P}(a_{\leq t}|o_{<t})] - \mathbf{H}[\mathbf{Q}(o_{\leq t}|a_{\leq t})], \\ \mathbf{E}[\mathbf{r}_\mathbf{P}(\underline{ao}_{\leq t})] &= -\mathbf{H}[\mathbf{P}(a_{\leq t}|o_{<t})] - \mathbf{H}[\mathbf{Q}(o_{\leq t}|a_{\leq t})] \\ &\quad - \mathbf{KL}[\mathbf{Q}(o_{\leq t}|a_{\leq t}) \parallel \mathbf{P}(o_{\leq t}|a_{\leq t})], \\ \mathbf{E}[\mathbf{r}_\mathbf{Q}(\underline{ao}_{\leq t})] &= -\mathbf{H}[\mathbf{P}(a_{\leq t}|o_{<t})] - \mathbf{H}[\mathbf{Q}(o_{\leq t}|a_{\leq t})] \\ &\quad - \mathbf{KL}[\mathbf{P}(a_{\leq t}|o_{<t}) \parallel \mathbf{Q}(a_{\leq t}|o_{<t})], \end{aligned}$$

where \mathbf{r}_G , \mathbf{r}_P and \mathbf{r}_Q are the reward functions for G , P and Q respectively. Note that \mathbf{H} and \mathbf{KL} are the entropy and the relative entropy functionals as defined in the appendix.

Accordingly, the interaction system's expected reward is given by the negative sum of the entropies produced by the agent's action generation probabilities and the environment's observation generation probabilities. The agent's (actual) expected reward is given by the negative cross-entropy between the generative distribution G and the agent's distribution P . The discrepancy between the agent's and the interaction system's expected reward is given by the relative entropy between the two probability distributions. Since the relative entropy is positive, one has $\mathbf{E}[\mathbf{r}_G(\underline{a}\underline{o}_{\leq t})] \geq \mathbf{E}[\mathbf{r}_P(\underline{a}\underline{o}_{\leq t})]$. This term implies that the better the environment is "modeled" by the agent, the better its performance will be. In other words: the agent has to recognize the structure of the environment to be able to exploit it. The designer can directly increase the agent's expected performance by controlling the first and the last term. The middle term is determined by the environment and only indirectly controllable. Importantly, the terms are in general coupled and not independent: changing one might affect another. For example, the first term suggests that less stochastic policies improve performance, which is oftentimes the case. However, in the case of a game with mixed Nash equilibria the overall reward can increase for a stochastic policy, which means that the first term is compensated for by the third term. Given the expected rewards, we can easily calculate the expected utilities in terms of entropy rates.

Corollary 1. *Let (P, Q) be an interaction system. The expected utilities of G , P and Q are given by*

$$\begin{aligned}\mathbf{E}[\mathbf{U}_G] &= \mathbf{G}\mathbf{U}_P + \mathbf{G}\mathbf{U}_Q \\ \mathbf{E}[\mathbf{U}_P] &= \mathbf{G}\mathbf{U}_P + \mathbf{G}\mathbf{U}_Q + \mathbf{P}\mathbf{U}_P \\ \mathbf{E}[\mathbf{U}_Q] &= \mathbf{G}\mathbf{U}_P + \mathbf{G}\mathbf{U}_Q + \mathbf{P}\mathbf{U}_Q\end{aligned}$$

where $\mathbf{G}\mathbf{U}_P$, $\mathbf{G}\mathbf{U}_Q$ and $\mathbf{P}\mathbf{U}_P$ are entropy rates defined as

$$\begin{aligned}\mathbf{G}\mathbf{U}_P &\equiv -\frac{1}{t} \sum_{\tau=1}^t \mathbf{H}[\mathbf{P}(a_\tau | \underline{a}\underline{o}_{<\tau})] \\ \mathbf{P}\mathbf{U}_P &\equiv -\frac{1}{t} \sum_{\tau=1}^t \mathbf{KL}[\mathbf{Q}(o_\tau | \underline{a}\underline{o}_{<\tau} a_\tau) \| \mathbf{P}(o_\tau | \underline{a}\underline{o}_{<\tau} a_\tau)] \\ \mathbf{G}\mathbf{U}_Q &\equiv -\frac{1}{t} \sum_{\tau=1}^t \mathbf{H}[\mathbf{Q}(o_\tau | \underline{a}\underline{o}_{<\tau} a_\tau)] \\ \mathbf{P}\mathbf{U}_Q &\equiv -\frac{1}{t} \sum_{\tau=1}^t \mathbf{KL}[\mathbf{P}(a_\tau | \underline{a}\underline{o}_{<\tau}) \| \mathbf{Q}(a_\tau | \underline{a}\underline{o}_{<\tau})].\end{aligned}$$

This result is easily obtained by dividing the quantities in Theorem 3 by t and then applying the chain rule for entropies to break the rewards over full sequences into instantaneous rewards. Note that $\mathbf{G}\mathbf{U}_P$, $\mathbf{G}\mathbf{U}_Q$ are the contributions to the utility due the generation of interactions, and $\mathbf{P}\mathbf{U}_P$, $\mathbf{P}\mathbf{U}_Q$ are the contributions to the utility due to the prediction of interactions.

Examples

One of the most interesting aspects of the information-theoretic formulation of utility is that it can be applied both to control problems (where an agent acts in a non-adaptive environment) and to game theoretic problems (where two possibly adaptive agents interact). In the following we apply the proposed utility measures to two simple toy examples from these two areas. In the first example, an adaptive agent interacts with a biased coin (the non-adaptive agent) and tries to predict the next outcome of the coin toss, which is either 'Head' (H) or 'Tail' (T). In the second example two adaptive agents interact playing the *matching pennies* game. One player has to match her action with the other player (HH or TT), while the other player has to unmatched (TH or HT). All agents have the same sets of possible observations and actions which are the binary sets $\mathcal{O} = \{H, T\}$ and $\mathcal{A} = \{H, T\}$.

Example 1. The non-adaptive agent is a biased coin. Accordingly, the coin's action probability is given by its bias and was set to $\mathbf{Q}(o = H) = 0.9$. The coin does not have any biased expectations about its observations, so we set $\mathbf{Q}(a = H) = 0.5$. The adaptive agent is given by the Laplace agent whose expectations over observed coin tosses follows the predictive distribution $\mathbf{P}(o = H|t, n) = (n+1)/(t+2)$, where t is the number of coin tosses observed so far and n is the number of observed Heads. Based on this estimator the Laplace agent chooses its action deterministically according to $\mathbf{P}(a = H|t, n) = \Theta(\frac{n+1}{t+2} - \frac{1}{2})$, where $\Theta(\cdot)$ is the Heaviside step function. From these distributions the full probability over interaction sequences can be computed. Figure 1A shows the entropy dynamics for a typical single run. The Laplace agent learns the distribution of the coin tosses, i.e. the KL decreases to zero. The negative cross-entropy stabilizes at the value of the observation entropy that cannot be further reduced. The entropy dynamics of the coin do not show any modulation.

Example 2. The two agents are modeled based on *smooth fictitious play* [Fudenberg and Kreps, 1993]. Both players keep count of the empirical frequencies of Head and Tail respectively. Therefore, each player i stores the quantities $\kappa_i^{(1)} = n_i$ and $\kappa_i^{(2)} = t - n_i$ where t is the number of moves observed so far, n_1 is the number of Heads observed by Player 1 and n_2 is the number of Heads observed by Player 2. The probability distributions $\mathbf{P}(o = H|t, n_1) = \gamma_1$ and $\mathbf{Q}(a = H|t, n_2) = \gamma_2$ over inputs is given by these empirical frequencies through $\gamma_i = \kappa_i / \sum_i \kappa_i$. The action probabilities are computed according to a sigmoid best-response function $\mathbf{P}(a = H|t, n_1) = 1/(1 + \exp(-\alpha(\gamma_1 - 0.5)))$, and $\mathbf{Q}(o = H|t, n_2) = 1/(1 + \exp(-\alpha(0.5 - \gamma_2)))$ respectively in case of Player 2 that has to unmatched. This game has a well-known equilibrium solution that is a mixed strategy Nash equilibrium where both players act

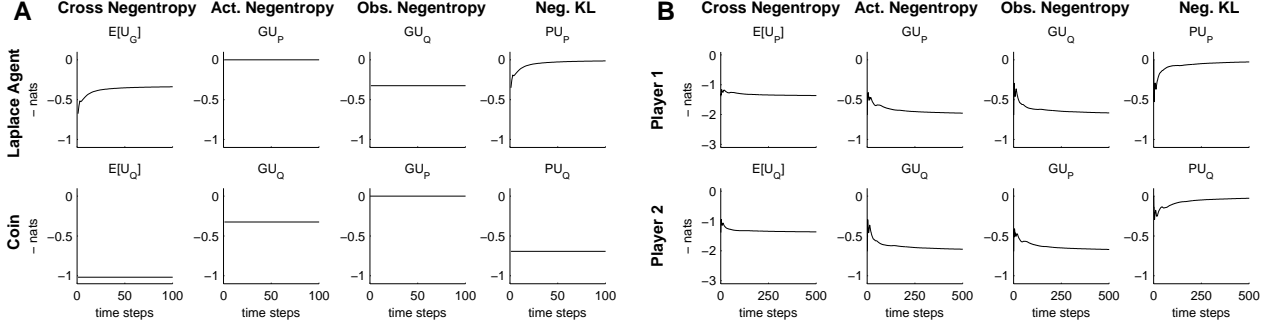


Figure 1: (A) Entropy dynamics of a Laplace agent interacting with a coin of bias 0.9. The Laplace agent learns to predict the coin’s behavior as can be seen in the decrease of the KL-divergence and the cross entropy. Since the Laplace agent acts deterministically its action entropy is always zero. Its observation entropy equals the action entropy of the coin. The coin does not change its behavior, which can be seen from the flat entropy curves. (B) Entropy dynamics of two adaptive agents playing matching pennies. Both agents follow smooth fictitious play. They converge to uniform random policies, which means that their action negentropies converge to $\log(2)$. Both agents learn the probability distribution of the other agent, as can be seen in the decrease of the KL-divergences.

randomly. Both action and observation entropies converge to the value $\log(2)$. Interestingly, the information-theoretic utility as computed by the cross-entropy takes the action entropy into account. Compare Figure 1B.

Conclusion

Based on three simple desiderata we propose that rewards can be measured in terms of information content and that, consequently, the entropy satisfies properties characteristic of a utility function. Previous theoretical studies have reported structural similarities between entropy and utility functions, see e.g. [Candea et al., 2001], and recently, relative entropy has even been proposed as a measure of utility in control systems [Todorov, 2009, Kappen et al., 2009, Ortega and Braun, 2008]. The contribution of this paper is to derive axiomatically a precise relation between rewards and information value and to apply it to coupled I/O systems.

The utility functions that we have derived can be conceptualized as *path utilities*, because they assign a utility value to an entire history. This is very similar to the path integral formulation in quantum mechanics where the utility of a path is determined by the classic action integral and the probability of a path is also obtain by taking the exponential of this ‘utility’ [Feynman and Hibbs, 1965]. In particular, we obtain the (cumulative time-averaged) cross entropy as a utility function when an agent is coupled to an environment. This utility function not only takes into account the KL-divergence as a measure of learning, but also the action entropy. This is interesting, because in most control problems controllers are designed to be deterministic (e.g. optimal control theory) in response to a known and stationary environment. If, however, the environment is not stationary and in fact adaptive as well, then it is a well-known result from game theory

that optimal strategies might be randomized. The utility function that we are proposing might indeed allow quantifying a trade-off between reducing the KL and reducing the action entropy. In the future it will therefore be interesting to investigate this utility function in more complex interaction systems.

Appendix

Entropy functionals

Entropy: Let \mathbf{Pr} be a probability distribution over $\mathcal{X} \times \mathcal{Y}$. Define the (average conditional) entropy [Shannon, 1948] as

$$\mathbf{H}[\mathbf{Pr}(x|y)] \equiv - \sum_{x,y} \mathbf{Pr}(x,y) \ln \mathbf{Pr}(x|y).$$

Relative Entropy: Let \mathbf{Pr}_1 and \mathbf{Pr}_2 be two probability distributions over $\mathcal{X} \times \mathcal{Y}$. Define the (average conditional) relative entropy [Kullback and Leibler, 1951] as

$$\mathbf{KL}[\mathbf{Pr}_1(x|y) \parallel \mathbf{Pr}_2(x|y)] \equiv \sum_{x,y} \mathbf{Pr}_1(x,y) \ln \frac{\mathbf{Pr}_1(x|y)}{\mathbf{Pr}_2(x|y)}.$$

Proof of Theorem 1

Proof. Let the function g be such that $g(\mathbf{Pr}(x)) = \mathbf{r}(x)$. Let $x_1, x_2, \dots, x_n \in \mathcal{F}$ be a sequence of events, such that $\mathbf{Pr}(x_1) = \mathbf{Pr}(x_i | x_{<i}) > 0$ for all $i = 2, \dots, n$. We have $\mathbf{Pr}(x_1, \dots, x_n) = \prod_i \mathbf{Pr}(x_i | x_{<i}) = \mathbf{Pr}(x_1)^n$. Since $\mathbf{Pr}(x) > \mathbf{Pr}(x') \Leftrightarrow \mathbf{r}(x) > \mathbf{r}(x')$ for any $x, x' \in \mathcal{F}$, then $\mathbf{Pr}(x) = \mathbf{Pr}(x') \Leftrightarrow \mathbf{r}(x) = \mathbf{r}(x')$, and thus $\mathbf{Pr}(x_1) = \mathbf{Pr}(x_i | x_{<i}) \Leftrightarrow \mathbf{r}(x_1) = \mathbf{r}(x_i | x_{<i})$ for all $i = 2, \dots, n$. This means, $\mathbf{r}(x_1, \dots, x_n) = n\mathbf{r}(x_1)$. But $g(\mathbf{Pr}(x_1, \dots, x_n)) = \mathbf{r}(x_1, \dots, x_n)$, and hence $g(\mathbf{Pr}(x_1)^n) = n\mathbf{r}(x_1)$. Similarly, for a second sequence of events $y_1, y_2, \dots, y_m \in \mathcal{F}$ with $\mathbf{Pr}(y_1) = \mathbf{Pr}(y_i | y_{<i}) > 0$ for all $i = 2, \dots, m$, we have $g(\mathbf{Pr}(y_1)^m) = m\mathbf{r}(y_1)$.

The rest of the argument parallels Shannon's entropy theorem [Shannon, 1948]. Define $p = \mathbf{Pr}(x_1)$ and $q = \mathbf{Pr}(y_1)$. Choose n arbitrarily high to satisfy $q^m \leq p^n < q^{m+1}$. Taking the logarithm, and dividing by $n \log q$ one obtains

$$\frac{m}{n} \leq \frac{\log p}{\log q} < \frac{m}{n} + \frac{1}{n} \Leftrightarrow \left| \frac{m}{n} - \frac{\log p}{\log q} \right| < \varepsilon,$$

where $\varepsilon > 0$ is arbitrarily small. Similarly, using $g(p^n) = n g(p)$ and the monotonicity of g , we can write $m g(q) \leq n g(p) < (m+1) g(q)$ and thus

$$\frac{m}{n} \leq \frac{g(p)}{g(q)} < \frac{m}{n} + \frac{1}{n} \Leftrightarrow \left| \frac{m}{n} - \frac{g(p)}{g(q)} \right| < \varepsilon,$$

where $\varepsilon > 0$ is arbitrarily small. Combining these two inequalities, one gets

$$\left| \frac{\log p}{\log q} - \frac{g(p)}{g(q)} \right| < 2\varepsilon,$$

which, fixing q , gives $\mathbf{r}(p) = g(p) = k \log p$, where $k > 0$. This holds for any $x_1 \in \mathcal{F}$ with $\mathbf{Pr}(x_1) > 0$. \square

Proof of Theorem 2

Proof. For all $\omega, \omega' \in \Omega$, $d(\omega) > d(\omega') \Leftrightarrow \alpha d(\omega) + \beta > \alpha d(\omega') + \beta \Leftrightarrow \mathbf{r}(\{\omega\}) > \mathbf{r}(\{\omega'\})$ because the affine transformation is positive. Now, the induced probability over $\mathcal{P}(\Omega)$ has atoms $\{\omega\}$ with probabilities $\mathbf{Pr}(\{\omega\}) = e^{\mathbf{r}(\{\omega\})} \geq 0$ and is normalized:

$$\sum_{\omega \in \Omega} e^{\mathbf{r}(\{\omega\})} = \sum_{\omega \in \Omega} e^{\alpha d(\{\omega\}) + \beta} = \frac{\sum_{\omega \in \Omega} e^{\alpha d(\omega)}}{\sum_{\omega \in \Omega} e^{\alpha d(\omega)}} = 1.$$

Since knowing $\mathbf{Pr}(\{\omega\})$ for all $\omega \in \Omega$ determines the measure for the whole field $\mathcal{P}(\Omega)$, $(\Omega, \mathcal{P}(\Omega), \mathbf{Pr})$ is a probability space. \square

Proof of Proposition 2

Proof. (i) Since $-\infty < \mathbf{r}(x_\tau | x_{<\tau}) \leq 0$ for all τ , then $-\infty < \frac{1}{t} \sum_{\tau=1}^t \mathbf{r}(x_\tau | x_{<\tau}) = \mathbf{U}(x_{\leq t}) \leq 0$ for all t . (ii) Write $\mathbf{Pr}(x_{\leq t})$ as

$$\begin{aligned} \mathbf{Pr}(x_{\leq t}) &= \prod_{\tau=1}^t \mathbf{Pr}(x_\tau | x_{<\tau}) = \prod_{\tau=1}^t \exp(\mathbf{r}(x_\tau | x_{<\tau})) \\ &= \exp\left(\sum_{\tau=1}^t \mathbf{r}(x_\tau | x_{<\tau})\right) = \exp(t \cdot \mathbf{U}(x_{\leq t})). \end{aligned}$$

(iii) $\mathbf{E}[\mathbf{U}(x_{\leq t})] = \sum_{x_{\leq t}} \mathbf{Pr}(x_{\leq t}) \mathbf{U}(x_{\leq t}) = \sum_{x_{\leq t}} \mathbf{Pr}(x_{\leq t}) \frac{1}{t} \mathbf{r}(x_{\leq t}) = -\frac{1}{t} \mathbf{H}[\mathbf{Pr}(x_{\leq t})]$, where we have applied (ii) in the second equality and $\mathbf{r}(\cdot) = \ln(\mathbf{Pr}(\cdot))$ in the third equality. \square

Proof of Theorem 3

Proof. This proof is done by straightforward calculation. First note that

$$\begin{aligned} \mathbf{G}(\underline{a}\underline{o}_{\leq t}) &= \prod_{\tau=1}^t \mathbf{P}(a_\tau | \underline{a}\underline{o}_{<\tau}) \mathbf{Q}(o_\tau | \underline{a}\underline{o}_{<\tau} a_\tau) \\ &= \mathbf{P}(a_{\leq t} | o_{<t}) \mathbf{Q}(o_{\leq t} | a_{\leq t}), \end{aligned}$$

which is obtained by applying multiple times the chain rule for probabilities and noting that the probability of a symbol is fully determined by the previous symbols. Similarly $\mathbf{P}(\underline{a}\underline{o}_{\leq t}) = \mathbf{P}(a_{\leq t} | o_{<t}) \mathbf{P}(o_{\leq t} | a_{\leq t})$ is obtained. We calculate here $\mathbf{E}[\mathbf{r}_\mathbf{P}(\underline{a}\underline{o}_{\leq t})]$. The calculation for $\mathbf{E}[\mathbf{r}_\mathbf{G}(\underline{a}\underline{o}_{\leq t})]$ and $\mathbf{E}[\mathbf{r}_\mathbf{Q}(\underline{a}\underline{o}_{\leq t})]$ are omitted because they are analogous.

$$\begin{aligned} \mathbf{E}[\mathbf{r}_\mathbf{P}(\underline{a}\underline{o}_{\leq t})] &\stackrel{(a)}{=} \sum_{\underline{a}\underline{o}_{\leq t}} \mathbf{G}(\underline{a}\underline{o}_{\leq t}) \ln \mathbf{P}(\underline{a}\underline{o}_{\leq t}) \\ &\stackrel{(b)}{=} \sum_{\underline{a}\underline{o}_{\leq t}} \mathbf{G}(\underline{a}\underline{o}_{\leq t}) \left(\ln \mathbf{P}(a_{\leq t} | o_{<t}) + \ln \mathbf{P}(o_{\leq t} | a_{\leq t}) \right) \\ &\stackrel{(c)}{=} \sum_{\underline{a}\underline{o}_{\leq t}} \mathbf{G}(\underline{a}\underline{o}_{\leq t}) \left(\ln \mathbf{P}(a_{\leq t} | o_{<t}) + \ln \mathbf{P}(o_{\leq t} | a_{\leq t}) \right. \\ &\quad \left. + \ln \mathbf{Q}(o_{\leq t} | a_{\leq t}) - \ln \mathbf{Q}(o_{\leq t} | a_{\leq t}) \right) \\ &\stackrel{(d)}{=} -\mathbf{H}[\mathbf{P}(a_{\leq t} | o_{<t})] - \mathbf{H}[\mathbf{Q}(o_{\leq t} | a_{\leq t})] \\ &\quad - \mathbf{KL}[\mathbf{Q}(o_{\leq t} | a_{\leq t}) \| \mathbf{P}(o_{\leq t} | a_{\leq t})]. \end{aligned}$$

Equality (a) follows from the definition of expectations and the relation between rewards and probabilities. In (b) we separate the term in the logarithm into the action and observation part. In (c) we add and subtract the term $\mathbf{Q}(o_{\leq t} | a_{\leq t})$ in the logarithm. Equality (d) follows from the algebraic manipulation of the terms and from identifying the entropy terms, noting that $\mathbf{G}(\underline{a}\underline{o}_{\leq t}) = \mathbf{P}(a_{\leq t} | o_{<t}) \mathbf{Q}(o_{\leq t} | a_{\leq t})$. \square

References

- J.C. Candeal, J.R. De Miguel, E. Induráin, and G.B. Mehta. Utility and entropy. *Economic Theory*, 17:233–238, 2001.
- R.P. Feynman and A.R. Hibbs. *Quantum Mechanics and Path Integrals*. McGraw-Hill, 1965.
- P.C. Fishburn. *The Foundations of Expected Utility*. D. Reidel Publishing, Dordrecht, 1982.
- D. Fudenberg and D.M. Kreps. Learning mixed equilibria. *Games and Economic Behavior*, 5:320–367, 1993.
- B. Kappen, V. Gomez, and M. Oppen. Optimal control as a graphical model inference problem. *arXiv:0901.0633*, 2009.
- S. Kullback and R.A. Leibler. On information and sufficiency. *The Annals of Mathematical Statistics*, 22(1):79–86, mar 1951. ISSN 0003-4851.
- D.J.C. MacKay. *Information Theory, Inference, and Learning Algorithms*. Cambridge University Press, 2003.
- P.A. Ortega and D.A. Braun. A minimum relative entropy principle for learning and acting. *arXiv:0810.3605*, 2008.
- S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice-Hall, Englewood Cliffs, NJ, 1st edition edition, 1995.
- C. E. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27:379–423 and 623–656, Jul and Oct 1948.
- E. Todorov. Efficient computation of optimal actions. *Proceedings of the National Academy of Sciences U.S.A.*, 106: 11478–11483, 2009.

Artificial General Segmentation

Daniel Hewlett and Paul Cohen

Department of Computer Science
University of Arizona
Tucson, AZ 85721, USA

Abstract

We argue that the ability to find meaningful chunks in sequential input is a core cognitive ability for artificial general intelligence, and that the Voting Experts algorithm, which searches for an information theoretic signature of chunks, provides a general implementation of this ability. In support of this claim, we demonstrate that VE successfully finds chunks in a wide variety of domains, solving such diverse tasks as word segmentation and morphology in multiple languages, visually recognizing letters in text, finding episodes in sequences of robot actions, and finding boundaries in the instruction of an AI student. We also discuss further desirable attributes of a general chunking algorithm, and show that VE possesses them.

Introduction

To succeed, artificial general intelligence requires domain-independent models and algorithms that describe and implement the fundamental components of cognition. Chunking is one of the most general and least understood phenomena in human cognition. George Miller described chunking as “a process of organizing or grouping the input into familiar units or chunks.” Other than being “what short term memory can hold 7 +/- 2 of,” chunks appear to be incommensurate in most other respects. Miller himself was perplexed because the information content of chunks is so different. A telephone number, which may be two or three chunks long, is very different from a chessboard, which may also contain just a few chunks but is vastly more complex. Chunks contain other chunks, further obscuring their information content. The psychological literature describes chunking in many experimental situations (mostly having to do with long-term memory) but it says nothing about the intrinsic, mathematical properties of chunks. The cognitive science literature discusses algorithms for forming chunks, each of which provides a kind of explanation of why some chunks rather than others are formed, but there are no explanations of what these algorithms, and thus the chunks they find, have in common.

The Signature of Chunks

Miller was close to the mark when he compared bits with chunks. Chunks may be identified by an information theoretic signature. Although chunks may contain vastly dif-

ferent amounts of Shannon information, they have one thing in common: Entropy within a chunk is relatively low, entropy at chunk boundaries is relatively high. Two kinds of evidence argue that this signature of chunks is general for the task of chunking sequences and series (see (KB01) for a similar idea applied to two-dimensional images). First, the Voting Experts (VE) chunking algorithm and its several variants, all of which detect this signature of chunks, perform very well in many domains. Second, when sequences are chunked all possible ways and ranked by a “chunkiness score” that combines within- and between-chunk entropy, the highest-ranked chunks are almost always real chunks according to a gold standard. Here, we focus primarily on the former kind of evidence, but also provide some early evidence of the latter kind.

Voting Experts

What properties should a general-purpose chunking algorithm have? It must not simply exploit prior knowledge of a particular domain, but rather must be able to learn to chunk novel input. It must operate without supervision in novel domains, and automatically set any parameters it has to appropriate values. For both humans and artificial agents, working memory is finite, and decisions must be made online, so the algorithm must be efficient and rely on local information rather than global optimization. Finally, learning should be rapid, meaning that the algorithm should have relatively modest data requirements.

VE has these properties. Its name refers to the “experts” that vote on possible boundary locations. The original version of VE had two experts: One votes to place boundaries after sequences that have low internal entropy, given by $H_I(seq) = -\log(p(seq))$, the other places votes after sequences that have high boundary entropy, given by $H_B(seq) = -\sum_{c \in S} p(c|seq) \log(p(c|seq))$, where S is the set of successors to seq . All sequences are evaluated locally, within a sliding window, so the algorithm is very efficient.

The statistics required to calculate H_I and H_B are stored efficiently using an n-gram trie, which is constructed in a single pass over the corpus. The trie depth is 1 greater than the size of the sliding window. Importantly, all statistics in the trie are normalized so as to be expressed in standard deviation units. This allows statistics from sequences of different lengths to be compared to one another.

The sliding window is passed over the corpus and each expert votes once per window for the boundary location that best matches its criteria. VE creates an array of vote counts, each element of which represents a location and the number of times an expert voted to segment at that location. The result of voting on the string `thisisacat` could be represented as `t0h0i1s3i1s4a4c1a0t`, where the numbers between letters are the total votes cast to split at the corresponding locations.

With vote totals in place, VE segments at locations that meet two requirements: First, the number of votes must be locally maximal (this is called the *zero crossing rule*). Second, the number of votes must exceed a *threshold*. Thus, VE has three parameters: the window size, the vote threshold, and whether to enforce the zero crossing rule. For further details of the VE algorithm see Cohen et al. (CAH07), and also Miller and Stoytchev (MS08). A fully-unsupervised version to the algorithm, which sets its own parameters, is described briefly later in the paper.

Extensions to Voting Experts

Some of the best unsupervised sequence-segmentation results in the literature come from the family of algorithms derived from VE. At an abstract level, each member of the family introduces an additional expert that refines or generalizes the boundary information produced by the two original VE experts to improve segmentation quality. Extensions to VE include Markov Experts (CM05), Hierarchical Voting Experts - 3 Experts (HVE-3E) (MS08), and Bootstrap Voting Experts (BVE) (HC09).

The first extension to VE introduced a “Markov Expert,” which treats the segmentation produced by the original experts as a data corpus and analyzes suffix/prefix distributions within it. Boundary insertion is then modeled as a Markov process based on these gathered statistics. HVE-3E is simpler: The third expert votes whenever it recognizes an entire chunk found by VE on the first iteration.

The new expert in BVE is called the *knowledge expert*. The knowledge expert has access to a trie (called the *knowledge trie*) that contains boundaries previously found by the algorithm, and votes to place boundaries at points in the sequence that are likely to be boundaries given this information. In an unsupervised setting, BVE generates its own supervision by applying the highest possible confidence threshold to the output of VE, thus choosing a small, high-precision set of boundaries. After this first segmentation, BVE repeatedly re-segments the corpus, each time constructing the knowledge trie from the output of the previous iteration, and relaxing the confidence threshold. In this way, BVE starts from a small, high-precision set of boundaries and grows it into a larger set with higher recall.

Related Algorithms

While Cohen and Adams (CA01) were the first to formulate the information-theoretic signature of chunks that drives VE, similar ideas abound. In particular, simpler versions of the chunk signature have existed within the morphology domain for some time.

Tanaka-Ishii and Jin (TIJ06) developed an algorithm called Phoneme to Morpheme (PtM) to implement ideas originally developed by Harris (Har55) in 1955. Harris noticed that if one proceeds incrementally through a sequence of phonemes and asks speakers of the language to list all the letters that could appear next in the sequence (today called the *successor count*), the points where the number *increases* often correspond to morpheme boundaries. Tanaka-Ishii and Jin correctly recognized that this idea was an early version of boundary entropy, one of the experts in VE. They designed their PtM algorithm based on boundary entropy in both directions (not merely the forward direction, as in VE), and PtM was able to achieve scores similar to those of VE on word segmentation in phonetically-encoded English and Chinese. PtM can be viewed as detecting an information-theoretic signature similar to that of VE, but relying only on boundary entropy and detecting change-points in the absolute boundary entropy, rather than local maxima in the standardized entropy.

Also within the morphology domain, Johnson and Martin’s HubMorph algorithm (JM03) constructs a trie from a set of words, and then converts it into a DFA by the process of minimization. Within this DFA, HubMorph searches for *stretched hubs*, which are sequences of states in the DFA that have a low branching factor internally, and high branching factor at the edges (shown in Figure 1). This is a nearly identical chunk signature to that of VE, only with successor/predecessor count approximating boundary entropy. The generality of this idea was not lost on Johnson and Martin, either: Speaking with respect to the morphology problem, Johnson and Martin close by saying “We believe that hub-automata will be the basis of a general solution for Indo-European languages as well as for Inuktitut.”

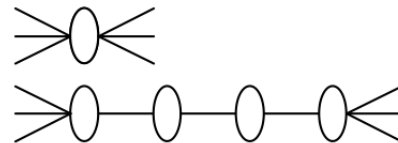


Figure 1: The DFA signature of a *hub* (top) and *stretched hub* in the HubMorph algorithm. Figure from Johnson and Martin.

VE Domains

To demonstrate the domain-independent chunking ability of VE, we now survey a variety of domains to which VE has been successfully. Some of these results appear in the literature, others are new and help to explain previous results. Unless otherwise noted, segmentation quality is measured by the boundary F-measure: $F = (2 \times \text{Precision} \times \text{Recall}) / (\text{Precision} + \text{Recall})$, where precision is the percentage of the induced boundaries that are correct, and recall is the percentage of the correct boundaries that were induced.

Language

VE and its variants have been tested most extensively in linguistic domains. Language arguably contains many levels of chunks, with the most natural being the word. The word segmentation task also benefits from being easily explained, well-studied, and having a large amount of gold-standard data available. Indeed, any text can be turned into a corpus for evaluating word segmentation algorithms simply by removing the word boundaries.

Word Segmentation Results for one corpus, in particular, have been reported in nearly every VE-related paper, and so is the most general comparison that can be drawn. This corpus is the first 50,000 characters of George Orwell’s *1984*. Table 1 shows the aggregated results for VE and its derivatives, as well as PtM.

Algorithm	Precision	Recall	F-score
VE	0.817	0.731	0.772
BVE	0.840	0.828	0.834
HVE-3E	0.800	0.769	0.784
Markov Exp.	0.809	0.787	0.798
PtM	0.766	0.748	0.757
All Points	0.185	1.000	0.313

Table 1: Results for VE and VE variants for word segmentation on an English text, *1984*.

Similar results can be obtained for different underlying languages, as well as different writing systems. Hewlett and Cohen showed similar scores for VE in Latin (F=0.772) and German (F=0.794) texts, and also presented VE results for word segmentation in orthographic Chinese (“Chinese characters”). VE achieved an F-score of 0.865 on a 100,000 word section of the Chinese Gigaword Corpus.

The higher score for Chinese than for the other languages has a simple explanation: Chinese characters correspond roughly to syllable-sized units, while the letters in the Latin alphabet correspond to individual phonemes. By grouping letters/phonemes into small chunks, the number of correct boundary locations remains constant, but the number of potential boundary locations is reduced. The means that even a baseline like All Locations, which places a boundary at every possible location, will perform better when segmenting a sequence of syllables than a sequence of letters.

VE has also been tested on phonetically-encoded English, in two areas: First, transcripts of of child-directed speech from the CHILDES database (MS85). Second, on a phonemic encoding of *1984* produced with the CMU pronouncing dictionary. On the CHILDES data, VE was able to find word boundaries as well or better (F=0.860) than several other algorithms, even though the other algorithms require their inputs to be sequences of utterances from which information about utterance beginnings and endings can be gathered (HC09). VE achieved an F-score of 0.807 on the phonemically-encoded version of *1984* (MS08).

Morphology While the word segmentation ability of VE has been studied extensively, its ability to find morphs has

not been examined previously. Morph segmentation is a harder task to evaluate than word segmentation, because intra-word morph boundaries are typically not indicated when writing or speaking. We constructed a gold standard corpus of Latin text segmented into morphs with the morphological analyzer *WORDS*.

Algorithm	Precision	Recall	F-score
PtM	0.630	0.733	0.678
VE	0.645	0.673	0.659
BidiVE	0.678	0.763	0.718
All Points	0.288	1.000	0.447

Table 2: Morph-finding results by algorithm. All Points is a baseline that places a boundary at every possible location.

From the table above (Table 2), it is clear that VE in its standard form has some difficulty finding the correct morphs. Still, its performance is comparable to PtM on this task, as expected due to the similarity in the two algorithms. PtM’s advantage probably is due to its bidirectionality: VE only actually examines the boundary entropy at the right (forward) boundary. VE was modified with the addition of an expert that places its votes *before* sequences that have high boundary entropy in the *backward* direction. This bidirectional version of VE, referred to as BidiVE, is a more faithful implementation of the idea that chunks are sequences with low internal entropy and high boundary entropy. BidiVE performed better than VE at finding morphs in Latin, as shown in the table.

For reference, when the task is to find word boundaries, the F-score for VE is approximately 0.77 on this same corpus. The reason for this is somewhat subtle: Because VE only looks at entropy in the *forward* direction, it will only consider the entropy after a morph, not before it. Consider a word like *senat.us*: The entropy of the next character following *senat* is actually fairly low, despite the fact that it is a complete morph. This is because the set of unique endings that can appear with a given stem like *senat* is actually fairly small, usually less than ten. Furthermore, in any particular text a word will only appear in certain syntactic relationships, meaning the set of endings it actually takes will be smaller still. However, the entropy of the character *preceding us* is very high, because *us* appears with a large number of stems. This fact goes unnoticed by VE.

Child Language Learning VE has also provided evidence relevant to an important debate within the child language learning literature: How do children learn to segment the speech stream into words? Famously, Saffran et al. (SAN96) showed that 8-month-old infants were able to distinguish correctly and incorrectly segmented words, even when those words were nonsense words heard only as part of a continuous speech stream. This result challenges models of word segmentation, such as Brent’s MBDP-1 (Bre99), which cannot operate without some boundary information. Saffran et al. proposed that children might segment continuous sequences at points of low transitional probability (TP), the simplest method which would successfully segment their

data.

However, TP alone performs very poorly on natural language, a fact which has not escaped opponents of the view that word segmentation is driven by distributional properties rather than innate knowledge about language. Linguistic nativists such as Gambell and Yang (GY05), argue that this failure of TP to scale up to natural language suggests that the statistical segmentation ability that children possess is limited and likely orthogonal to a more powerful segmentation ability driven by innate linguistic knowledge. Gambell and Yang demonstrate that an algorithm based on linguistic constraints (specifically, constraints on the pattern of syllable stress in a word) significantly outperforms TP when segmenting a corpus of phonetically-encoded child-directed speech. In fact, VE can further outperform Gambell and Yang's method ($F=0.953$ vs. $F=0.946$) even though VE has no prior knowledge of linguistic constraints, suggesting that adding innate knowledge may not be as useful as simply increasing the power of the chunking method.

Algorithms like VE and PtM provide a counter-argument to the nativist position, by fully explaining the results that Saffran et al. observed, and also performing very well at segmenting natural language. When represented symbolically as a sequence of phonemes, VE perfectly segments the simple artificial language generated by Saffran et al. (SAN96), while also performing well in the segmentation of child-directed speech. Miller et al. (MWS09) reinforce this case by replicating the experimental setup of Saffran et al., but feeding the speech input to VE instead of a child. The audio signal had to be discretized before VE could segment it, but VE was able to achieve an accuracy of 0.824.

Vision

Miller and Stoytchev (MS08) applied VE in a hierarchical fashion to perform a visual task similar to optical character recognition (OCR). The input was an image containing words written in a particular font. VE was to first segment this image into short sequences corresponding to letters, and then chunk the short sequences into longer sequences corresponding to words. The image was represented as a sequence of columns of pixels, where each pixel was either black or white. Each of these pixel columns can be represented by a symbol denoting the particular pattern of black and white pixels within it, thus creating a sequence of symbols to serve as input to VE. Depending on the font used, VE scored between $F=0.751$ and $F=0.972$ on segmenting this first sequence.

After finding letters, VE had to chunk these letters together into words, which is essentially the same as the well-studied word segmentation problem except with some noise added to the identification of each character. VE was still able to perform the task, with scores ranging from $F=0.551$ to $F=0.754$ for the three fonts. With perfect letter identification, VE scored $F=0.776$.

Robot Behaviors

Cohen et al. (CAH07) tested VE on data generated by a mobile robot, a Pioneer 2 equipped with sonar and a pan-tilt-zoom camera running a subsumption architecture. The

robot wandered around a large playpen for 20-30 minutes looking for interesting objects, which it would orbit for a few minutes before moving on. At one level of abstraction, the robot engaged in four types of behaviors: wandering, avoiding, orbiting and approaching. Each behavior was implemented by sequences of actions initiated by controllers such as move-forward and center-camera-on-object. The challenge for Voting Experts was to find the boundaries of the four behaviors given only information about which controllers were on or off.

This experiment told us that the encoding of a sequence matters: When the coding produced shorter behaviors (average length of 7.95 time steps), VE's performance was comparable to that in earlier experiments ($F=0.778$), but when the coding produced longer behaviors, performance is very much worse ($F=0.183$). This is because very long episodes are unique, so most locations in very long episodes have zero boundary entropy and frequency equal to one. And when the window size is very much smaller than the episode length, then there will be a strong bias to cut the sequence inappropriately.

Instruction of an AI Student

The goal of the DARPA's Bootstrapped Learning (BL) project is to develop an "electronic student" that can be instructed by human teachers, in a natural manner, to perform complex tasks. Currently, interaction with the electronic student is not very different from high-level programming. Our goal is to replace many of the formal cues or "signposts" that enable the electronic student to follow the teacher, making the interaction between them more natural. VE can largely replace one of these cues: the need to inform the student whenever the teacher's instruction method changes.

In BL, teachers communicate with the student in a language called *Interlingua language* (IL). Some IL messages serve only to notify the student that a "Lesson Epoch" (LE) has ended.

Several curricula have been developed for BL. VE finds LE boundaries with high accuracy in all of them – and can be trained on one and tested on another to good effect. To illustrate, we will present results for the Unmanned Aerial Vehicle (UAV) domain. To study the detection of LE boundaries, a training corpus was generated from version 2.4.01 of the UAV curriculum by removing all of the messages that indicate boundaries between LEs. This training corpus contains a total of 742 LEs. A separate corpus consisting of 194 LEs served as a test corpus. As the teacher should never have to provide LE boundaries, the problem is treated as unsupervised and both the training and test corpora are stripped of all boundary information.

Each individual message in the corpus is a recursive structure of IL objects that together express a variety of relations about the concepts being taught and the state of teaching. LEs are defined more by the structure of the message sequence than the full content of each message. Thus, we represent each message as a single symbol, formed by concatenating the IL type of the two highest composite IL objects (generally equivalent to the message's type and subtype). The sequence of structured messages is thus translated into

Size	TRAINING			TEST		
	P	R	F	P	R	F
1.00	0.927	0.888	0.907	0.933	0.876	0.904
0.75	0.881	0.839	0.859	0.904	0.829	0.864
0.50	0.905	0.784	0.840	0.871	0.772	0.819
0.25	0.961	0.772	0.856	0.836	0.606	0.703

Table 3: BVE Results on UAV Domain trained on different subsets of the training corpus. “Size” is percentage of the training corpus given to BVE.

a sequence of symbols, and it is this symbol sequence that will be segmented into LEs.

BVE is allowed to process the training corpus repeatedly to gather statistics and segment it, but the segmentation of the test corpus must be done in one pass, to model more closely the constraints of a real teacher-student interaction. If allowed to operate on the full UAV corpus, BVE finds LE boundaries handily, achieving an F-score of 0.907. However, this domain is non-trivial: VE achieves an F-score of 0.753, only slightly lower than its score for word segmentation in English text. As a baseline comparison, segmenting the corpus at every location results in an F-score of 0.315, which indicates that LE boundaries are roughly as frequent as word boundaries in English, and thus that high performance is not guaranteed simply by the frequency of boundaries of the data.

Results from segmenting a test corpus (not drawn from the training corpus) consisting of 194 lesson epochs are shown in Table 3. “Training Size” refers to the percentage of the training corpus processed by BVE before segmenting the test corpus. From these results, it is evident that BVE can perform very well on a new corpus when the training corpus is sufficiently large. However, with a small training corpus BVE does not encounter certain boundary situations, and thus fails to recognize them during the test, resulting in lower recall.

Evidence for Generality

So far, we have discussed in detail one kind of evidence for the general applicability of VE, namely that VE successfully performs unsupervised segmentation in a wide variety of domains. In order for VE to be successful in a given domain, chunks must exist in that domain that adhere to the VE’s signature of chunks, and VE must correctly identify these chunks. Thus, the success of VE in each of these domains is evidence for the presence of chunks that adhere to the signature in each domain. Also, VE’s chunk signature is similar to (or a direct generalization of) several other independently-developed signatures, such as PtM, HubMorph, and the work of Kadir and Brady (KB01). The independent formulation of similar signatures by researchers working in different domains suggests that a common principle is at work across those domains.

Optimality of the VE Chunk Signature

Though the success of VE in a given domain provides indirect evidence that the chunk signature successfully identifies chunks in that domain, we can evaluate the validity of the chunk signature much more directly. To evaluate the ability of the chunk signature to select the true segmentation from among all possible segmentations of a given sequence, we developed a “chunkiness” score that can be assigned to each possible segmentation, thus ranking all possible segmentations by the quality of the chunks they contain. The chunkiness score rewards frequent sequences that have high entropy at both boundaries (Equation 1), just as in VE. The score for a complete segmentation is simply the average of the chunkiness of each segment. If the chunk signature is correct, the true segmentation should have a very high score, and so will appear close to the top of this ranking. Unfortunately, due to the exponential increase in the number of segmentations (a sequence of length n has 2^{n-1} segmentations), this methodology can only be reasonably applied to short sequences. However, it can be applied to many such short sequences to better gain a better estimate of the degree to which optimizing chunkiness optimizes segmentation quality.

$$Ch(s) = \frac{H_f(s) + H_b(s)}{2} - \log Pr(s) \quad (1)$$

For each 5-word sequence (usually between 18 and 27 characters long) in the Bloom73 corpus from CHILDES, we generated all possible segmentations and ranked them all by chunkiness. On average, the true segmentation was in the 98.7th percentile. All probabilities needed for computing the chunkiness score were estimated from a training corpus, the Brown73 corpus (also from CHILDES). Preliminarily, it appears that syntax is the primary reason that the true segmentation is not higher in the ranking: When the word-order in the training corpus is scrambled, the true segmentation is in the 99.6th percentile. Still, based on these early results we can say that, in at least one domain, optimizing chunkiness very nearly optimizes segmentation quality.

Automatic Setting of Parameters

VE has tunable parameters, and Hewlett and Cohen (HC09) showed that these parameters can greatly affect performance. However, they also demonstrated how these parameters can be tuned without supervision. Minimum Description Length (MDL) provides an unsupervised way to set these parameters indirectly by selecting among the segmentations each combination of parameters generates. The *Description Length* for a given hypothesis and data set refers to the number of bits needed to represent both the hypothesis and the data given that hypothesis. The *Minimum Description Length*, then, simply refers to the principle of selecting the hypothesis that minimizes description length. In this context, the data is a corpus (sequence of symbols), and the hypotheses are proposed segmentations of that corpus, each corresponding to a different combination of parameter settings. Thus, we choose the vector of parameter settings that generates the hypothesized segmentation which has the minimum description length.

Extension to Non-Symbolic Data

Strictly speaking, VE can only operate over sequences of discrete symbols. However, as already demonstrated by Miller et al.'s applications of VE to the visual and auditory domains, many sequences of multivariate or continuous-valued data can be transformed into a symbolic representation for VE. Also, the SAX algorithm (LKWL07) provides a general way to convert a stream of continuous data into a sequence of symbols.

Allowing Supervision

While the ability of VE to operate in a fully unsupervised setting is certainly a strength, the fact that VE contains no natural mechanism for incorporating supervision may be seen as a limitation: If some likely examples of ground truth boundaries are available, the algorithm ought to be able to take advantage of this information. While VE itself cannot benefit from true boundary knowledge, one of its extensions, BVE, does so handily. BVE's knowledge trie can store previously discovered boundaries (whether provided to or inferred by the algorithm), and the knowledge expert votes for boundary locations that match this prior knowledge. The Markov Experts version is able to benefit from supervision in a similar way, and, if entire correct chunks are known, HVE-3E can as well.

An Emergent Lexicon

VE does not represent explicitly a "lexicon" of chunks that it has discovered. VE produces chunks when applied to a sequence, but its internal data structures do not represent the chunks it has discovered explicitly. By contrast, BVE stores boundary information in the knowledge trie and refines it over time. Simply by storing the beginnings and endings of segments, the knowledge trie comes to store sequences like #cat#, where # represents a word boundary. The set of such bounded sequences constitutes a simple, but accurate, emergent lexicon. After segmenting a corpus of child-directed speech, the ten most frequent words of this lexicon are *you*, *the*, *that*, *what*, *is*, *it*, *this*, *what's*, *to*, and *look*. Of the 100 most frequent words, 93 are correct. The 7 errors include splitting off morphemes such as *ing*, and merging frequently co-occurring word pairs such as *do you*.

Conclusion

Chunking is one of the domain-independent cognitive abilities that is required for general intelligence, and VE provides a powerful and general implementation of this ability. We have demonstrated that VE and related algorithms perform well at finding chunks in a wide variety of domains, and provided preliminary evidence that chunks found by maximizing chunkiness are almost always real chunks. This suggests that the information theoretic chunk signature that drives VE is not specific to any one domain or small set of domains. We have discussed how extensions to VE enable it to operate over nearly any sequential domain, incorporate supervision when present, and tune its own parameters to fit the domain.

References

- [Bre99] Michael R Brent. An Efficient, Probabilistically Sound Algorithm for Segmentation and Word Discovery. *Machine Learning*, pages 71–105, 1999.
- [CA01] P Cohen and N Adams. An algorithm for segmenting categorical time series into meaningful episodes. *Lecture notes in computer science*, 2001.
- [CAH07] Paul Cohen, Niall Adams, and Brent Heeringa. Voting Experts: An Unsupervised Algorithm for Segmenting Sequences. *Intelligent Data Analysis*, 11:607–625, 2007.
- [CM05] Jimming Cheng and Michael Mitzenmacher. The Markov Expert for Finding Episodes in Time Series. In *Proceedings of the Data Compression Conference (DCC 2005)*, pages 454–454. IEEE, 2005.
- [GY05] Timothy Gambell and Charles Yang. Word Segmentation: Quick but not Dirty. 2005.
- [Har55] Zellig S. Harris. From Phoneme to Morpheme. *Language*, 31:190, 1955.
- [HC09] Daniel Hewlett and Paul Cohen. Bootstrap Voting Experts. In *Proceedings of the Twenty-first International Joint Conference on Artificial Intelligence (IJCAI-09)*, 2009.
- [JM03] Howard Johnson and Joel Martin. Unsupervised learning of morphology for English and Inuktitut. *Proceedings of the 2003 North American Chapter of the Association for Computational Linguistics on Human Language Technology (NAACL-HLT 03)*, pages 43–45, 2003.
- [KB01] Timor Kadir and Michael Brady. Saliency, Scale and Image Description. *International Journal of Computer Vision*, 45:83–105, 2001.
- [LKWL07] Jessica Lin, Eamonn Keogh, Li Wei, and Stefano Lonardi. Experiencing SAX: a novel symbolic representation of time series. *Data Mining and Knowledge Discovery*, 15:107–144, April 2007.
- [MS85] Brian McWhinney and Cynthia E. Snow. The child language data exchange system (CHILDES). *Journal of Child Language*, 1985.
- [MS08] Matthew Miller and Alexander Stoytchev. Hierarchical Voting Experts: An Unsupervised Algorithm for Hierarchical Sequence Segmentation. In *Proceedings of the 7th IEEE International Conference on Development and Learning (ICDL 2008)*, pages 186–191, 2008.
- [MWS09] Matthew Miller, Peter Wong, and Alexander Stoytchev. Unsupervised Segmentation of Audio Speech Using the Voting Experts Algorithm. *Proceedings of the 2nd Conference on Artificial General Intelligence (AGI 2009)*, 2009.
- [SAN96] Jenny R Saffran, Richard N Aslin, and Elissa L Newport. Statistical Learning by 8-Month-Old Infants. *Science*, 274:926–928, 1996.
- [TIJ06] Kumiko Tanaka-Ishii and Zhihui Jin. From Phoneme to Morpheme: Another Verification Using a Corpus. In *Proceedings of the 21st International Conference on Computer Processing of Oriental Languages (ICCPOL 2006)*, volume 4285, pages 234–244, 2006.

The CHREST Architecture of Cognition

The Role of Perception in General Intelligence

Fernand Gobet

Centre for the Study of Expertise
Brunel University
Uxbridge Middlesex, UB8 3PH
fernand.gobet@brunel.ac.uk

Peter C.R. Lane

School of Computer Science
University of Hertfordshire
Hatfield, Hertfordshire, AL10 9AB
peter.lane@bcs.org.uk

Abstract

This paper argues that the CHREST architecture of cognition can shed important light on developing artificial general intelligence. The key theme is that “cognition is perception.” The description of the main components and mechanisms of the architecture is followed by a discussion of several domains where CHREST has already been successfully applied, such as the psychology of expert behaviour, the acquisition of language by children, and the learning of multiple representations in physics. The characteristics of CHREST that enable it to account for empirical data include: self-organisation, an emphasis on cognitive limitations, the presence of a perception-learning cycle, and the use of naturalistic data as input for learning. We argue that some of these characteristics can help shed light on the hard questions facing theorists developing artificial general intelligence, such as intuition, the acquisition and use of concepts, and the role of embodiment.

Introduction

There are two main broad approaches for developing general artificial intelligence. The first is to use whatever techniques are offered by computer science and artificial intelligence, including brute force, to create artefacts that behave in an intelligent way. The second is to develop computational architectures that closely simulate human behaviour in a variety of domains. Examples of this approach include ACT-R (Anderson and Lebière, 1998), Soar (Newell, 1990), and EPAM (Feigenbaum and Simon, 1984). More recently, the computational architecture CHREST (Chunk Hierarchy and REtrieval Structures) (Gobet et al., 2001; Gobet and Simon, 2000; Lane, Cheng, and Gobet, 2000) has simulated data in a number of domains, including expert behaviour in board games, problem solving in physics, first language acquisition, and implicit learning.

The strength of cognitive architectures is that their implementation as computer programs ensures a high degree of precision, and offers a sufficiency proof that the mechanisms proposed can carry out the tasks under study – something obviously desirable if artificial general intelligence is the goal. The extent to which success is reached in simulating actual human behaviour can be assessed by using measures such as eye movements, reaction times, and error patterns.

The aim of this paper is to introduce CHREST, to

illustrate the kind of phenomena it has already been able to successfully simulate, and to show what insight it offers on the creation of AI systems displaying general intelligence. The claim made here is that developing a cognitive architecture – and thus understanding human intelligence – provides critical insight for developing general artificial intelligence.

The CHREST Architecture

Just like its predecessor, EPAM (Elementary Memorizer and Perceiver) (Feigenbaum and Simon, 1984), CHREST assumes the presence of short-term memory (STM) and long-term memory (LTM) structures, and models cognition as the product of the interaction of perceptual learning, memory retrieval, and decision-making processes. A central theme is that “cognition is perception” (De Groot and Gobet, 1996). Thus, the architecture postulates a close interaction between perception, learning, and memory: CHREST’s knowledge directs attention and perception, and, in turn, perception directs the learning of new knowledge.

Another essential aspect of the architecture is that Simon’s assumption of bounded rationality (Simon, 1969) is taken very seriously. For example, CHREST’s behaviour is constrained by the limited capacity of visual short-term memory (3 chunks), the relatively slow rate at which new elements can be learned (8 seconds to create a new chunk), and the time it takes to transfer information from LTM to STM (50 ms). Just like the human cognitive system, CHREST satisfices, and this might be a key condition for displaying general intelligence. All cognitive operations carried out by the system have a cost, which is indicated with approximate but fixed time parameters. The presence of these parameters enables close comparison between human and simulated behaviour (see De Groot and Gobet, 1996, for details; a technical specification of CHREST can be found at www.CHREST.info).

The emphasis on cognitive limitations is in stark contrast with architectures such as Soar, where the stress is on carrying out complex intelligent behaviour without imposing many constraints on the architecture (for example, Soar enjoys an unlimited capacity for its working memory). Compared to other architectures, CHREST might thus appear as a very austere system. However, it is also a powerful dynamic system governed not only by built-in capabilities

but also, more importantly, by the complexities of its interaction with the environment. Together, these features enable it to cover a wide range of behaviours.

Components

The three main components of CHREST are shown in Figure 1: (a) mechanisms and structures for interacting with the external world; (b) multiple STMs that hold information from varied input modalities; and (c) an LTM, which holds information in a “chunking network.” A chunking network is a discrimination network whose dynamic growth is a joint function of the previous states of the system and the inputs from the environment. The visual input-output channels (and the STM associated with them) have been investigated in models of chess expertise where the actual eye movements of masters and novices have been simulated (De Groot and Gobet, 1996). In general, this simulated eye is crucial in understanding the interaction between low-level information, such as visual stimuli, and high-level cognition, such as concepts. The auditory channels have been investigated in a model of vocabulary acquisition (Jones, Gobet and Pine, 2007, 2008), which simulates the way children learn words using phonemic information.

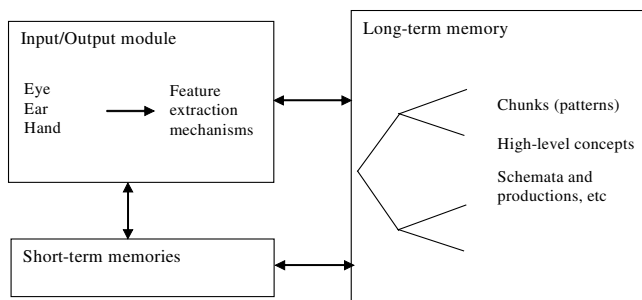


Figure 1. Key components of the CHREST architecture

Learning Mechanisms

Chunking networks are grown by mechanisms similar to those used in EPAM. A process called *discrimination* creates new nodes and a process called *familiarisation* incrementally adds information to existing nodes. An important extension, compared to EPAM, is a mechanism for creating high-level structures from perceptual information, by a process called *template formation*. The creation of a template, which is a kind of schema, uses both stable information (for creating its *core*) and variable information (for creating its *slots*). Templates are essential for explaining how chess Masters can recall briefly presented positions relatively well, even with a presentation time as short as 1 or 2 seconds (Gobet and Simon, 2000). They are also important for explaining how chess masters carry out planning – that is search at a level higher than that of moves. Another important novelty is the creation of *lateral links* (similarity links, production links, equivalence links, and generative links) between nodes (see Gobet et al., 2001, for detail). It is important to point out

that all these mechanisms are carried out automatically. Much more than with previous chunking models (for example, Simon and Gilmarin, 1973), CHREST explains the development of expertise by both acquiring a large number of knowledge structures and building connections between them.

Eye Movements and the Perception-Learning Cycle

The *frame problem* is a central issue in cognitive science and artificial intelligence: How can a system notice the relevant changes in the environment in real time whilst ignoring the indefinite number of changes that are irrelevant? CHREST’s solution consists of three parts, which all lead to a reduction in information. First, the limited capacity of the visual field eliminates a considerable amount of information coming from the environment. Second, the knowledge that the system brings to bear – and sometimes the lack of such knowledge – further constrains the amount of information processed. Third, the limited memory capacity we have mentioned earlier causes a further reduction of information. Thus, CHREST is highly selective, as presumably is the human cognitive system. This is consistent with research on perception and evolution, which has shown that animals and humans in particular have evolved powerful perceptual mechanisms for extracting key features from sensations in order to survive complex and dangerous environments.

Indeed, with CHREST, the link between perception and cognition is so tight that the distinction between these two sets of mechanisms all but disappears. To begin with, the focus of attention determines what information will be learned. Then, when possible, eye movements and thus attention will be directed by previous knowledge, making it more likely that the system pays attention to critical information. The key assumption here is that features that were important in the past – to the point that they led to learning – should be important in the future as well. This perception-learning-perception cycle is another way by which CHREST addresses the frame problem, as it leads to a selectivity of attention that further reduces the amount of information extracted from the environment and makes it possible to respond in real time.

Some Domains Modelled by CHREST

Chess Expertise

Historically, chess has been a standard domain for studying cognition, including intelligence, both for humans and computers. Chess was the first domain of application of CHREST, a domain that turned out to be excellent as it engages various cognitive abilities including perception, memory, decision making, and problem solving. It turns out that CHREST can simulate a large number of phenomena related to chess expertise. These include: the eye movements of novices and chess Masters (see Figure 2); recall performance in numerous memory experiments (including errors and the detail of the piece placements); and evolution

of look-ahead search as a function of skill (De Groot and Gobet, 1996; Gobet, 1997; Gobet and Simon, 1996, 2000). The main mechanism explaining these phenomena is the acquisition of a large number of chunks (more than 300,000 for simulating Grandmasters) and templates, which are autonomously acquired from scanning master-level games.

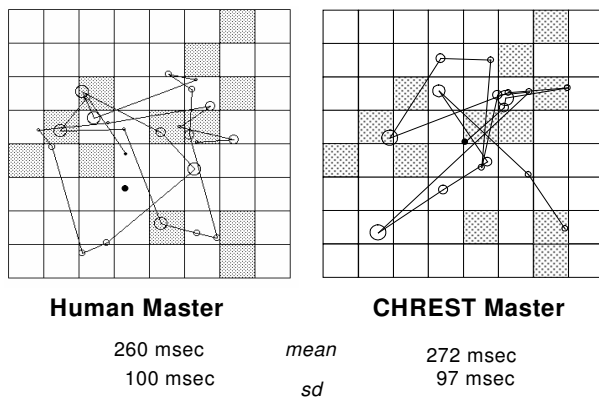


Figure 2. Example of a Master's eye movements (left) and their simulations by CHREST (right). The mean and the standard deviation of fixation times are across all positions and all master subjects. (After De Groot and Gobet, 1996.)

Computational modelling is often criticised by describing it as only some kind of curve-fitting exercise, without new predictions and thus without real understanding. CHREST does not suffer from this weakness, as it has made a number of new predictions, some of which have been later confirmed by empirical data. A good example of this is the recall of random chess positions, which are constructed by haphazardly replacing the pieces of a game position on the board. When these simulations were carried out, the universal belief, based on research by Chase and Simon (1973) with a small sample, was that there was no skill difference in the recall of random positions, while of course Masters vastly outperform weaker players with game positions. The simulations repeatedly showed that there should also be a skill effect with this type of positions – for the simple reason that CHREST could recognize some chunks in random positions, just by chance, and that this was more likely with large networks of chunks. This prediction was verified by a re-analysis of all studies carried out with this type of material and by collecting new data (Gobet and Simon, 1996). Figure 3 shows the predictions of CHREST and the human data, for both game and random positions. (Please note the close fit between CHREST and the human data.) CHREST's predictions were also supported when random positions were created using different procedures (Gobet and Waters, 2003).

These results indicate that Masters perceive patterns in spite of the fact that the positions do not contain much structure, a further indication, if necessary, that chunking mechanisms are automatic and implicit. If this is the case, CHREST should be able to simulate the kind of phenomena observed in the implicit learning literature (Reber, 1967). In

an implicit-learning experiment, stimuli generated from an artificial grammar are first shown to the subjects, and then subjects have to classify new strings as well-formed or not. The debate in this literature is whether subjects learn anything in these experiments, and, if so, what kind of things are learnt: rules, fragments, or exemplars? Unpublished work with Daniel Freudenthal indicates that CHREST can simulate some of the key results very well, suggesting that subjects in these experiments learn implicitly and unconsciously small fragments of stimuli that become incrementally larger with additional learning.

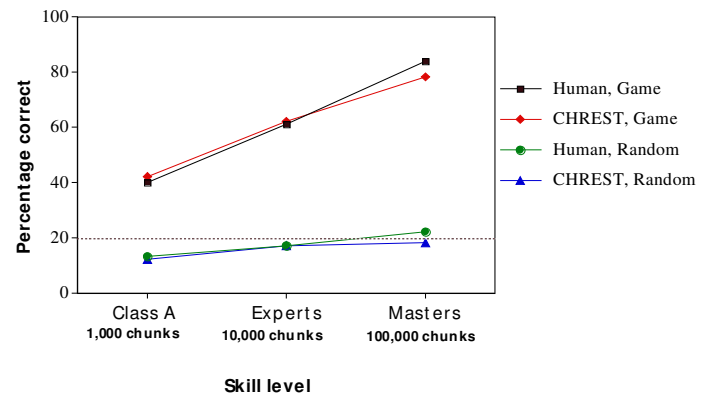


Figure 3. Memory for game and random positions as a function of skill level. The human data are the average results aggregated across 13 studies. (After Gobet and Simon, 1996.)

Of course, the essence of chess skill is not to memorize chess positions, but to find good moves. Work has also been done with CHREST to understand how human masters are able to find good moves despite searching only one very small subset of the search space. A first program inspired by CHREST, CHUMP (CHUnking of Moves and Patterns; Gobet and Jansen, 1994), was able to learn an association of chess moves to perceptual patterns. It is interesting that CHUMP, while playing at a fairly low level, performed better in positions requiring a 'positional judgement' than in tactical positions, which engage more look-ahead search. As positional judgment in chess is seen as a clear-cut example of intuition (e.g., Dreyfus & Dreyfus, 1986), it could be argued that CHUMP captures this aspect of human intelligence (see also Simon, 1979). A second program, SEARCH (Gobet, 1997), simulated key parameters of the way human of different levels search a position. These parameters included depth of search, width of search, and rate at which moves were generated. More recent work with CHREST has added mechanisms combining look-ahead search and pattern recognition, paving the way to complete simulation of expertise in chess.

Other Domains of Expertise

An application of CHREST to the African game of Awele (Gobet, 2009) shows that it can play at a fairly good level by

pattern recognition only, while at the same time simulating several results from memory-recall experiments. Simulations have also been carried out on memory for computer programs and the acquisition of multiple representations in physics. In this latter work, the aim was to study the acquisition of multiple diagrammatic representations, and the combination of these multiple representations to form problem-solving stages. The representations were, first, the classic circuit-style of representation, found in textbooks, and second, a specialized problem-solving representation, containing quantitative properties of the domain (see Lane et al., 2000 for details). The essential elements of the model here were the movements of the eye and visual short-term memory; some chunks were learnt for each representation, and they were combined within short-term memory using *lateral links*. These links are used in solving new problems, to retrieve known components of a problem.

Although simple in comparison to the abilities of ACT-R or Soar, this work provides CHREST with a rather unique form of problem solving, based around perceptual chunks. The idea is that problem solutions are planned, at a high level, by retrieving familiar chunks; these familiar chunks have been acquired by solving smaller problems in isolation. The solution process involves a form of composition, guided by the perceived separation of the problem into chunks. The decomposition used by CHREST corresponds with that used and drawn by human participants, providing empirical support for the role of perceptual chunks in problem solving.

Linking Perception to Expectations

Perception in CHREST is seen as a cycle, with the eye guided to look at parts of the scene or image where useful information is expected to lie. With human, and perhaps animal, cognition, it is expected that such expectations would be formed from information in more than one modality. For example, knowing a sequence of verbal statements may help guide the location of a sequence of perceptual icons.

Lane, Sykes and Gobet (2003) explored this aspect of cognition by training CHREST to encode information in more than one modality: visual and verbal. The process is illustrated by Figure 4. Here, one visual and one verbal stimulus are separately experienced and sorted through the long-term memory. Pointers to the retrieved chunks are placed into short-term memory. Because short-term memory is separated for each modality, the information about visual and verbal chunks is kept distinct. An association is then made between the visual and verbal memories.

The interaction between the two memories produces various measurable effects. For instance, prior expectations can improve the speed and accuracy with which unclear stimuli are recognised. The ultimate aim is to understand how low-level and high-level knowledge interact to produce intelligent behaviour, a question that has nagged cognitive science for decades (for example, Neisser, 1966).

Current work is looking to apply chunking mechanisms to bitmap-level data, whilst maintaining interactions between verbal and visuo-spatial information. One technique for doing so lies in employing techniques from component-based

vision, where one layer of feature detectors seeks out symbolic features to pass to a second layer. For example, Han et al. (2009) employ support-vector machines to locate components of a human face, which are then classified by a separate classifier. A natural extension is to use CHREST as the second layer, and so gain the benefits both of low-level pixel analysis and high-level symbolic pattern matching across multiple representations.

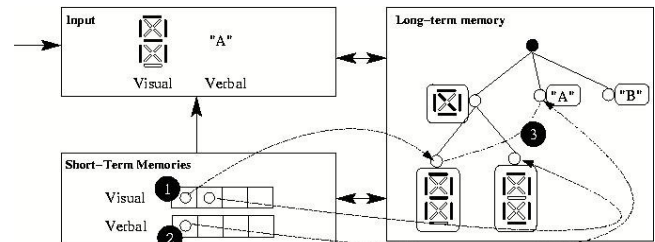


Figure 4. Learning to link information across two modalities. (1) The visual pattern is sorted through LTM, and a pointer to the node retrieved placed into visual STM. (2) The verbal pattern is sorted through LTM, and a pointer to the node retrieved placed into verbal STM. (3) A naming link is formed between the two nodes at the top of the STMs. (After Lane, Sykes and Gobet, 2003.)

Acquisition of Language

Language is uniquely human, and understanding how children acquire their first language will tell us something important about intelligence in general. We see language acquisition as one type of expertise, and argue that children become experts in their native language through implicitly acquiring a large number of chunks and links between them. A first variation of CHREST has studied the acquisition of vocabulary (Jones et al., 2007). The interest has been on how mechanisms in short-term memory and long-term memory interact through the creation and use of chunks. A second variation of CHREST, known as MOSAIC (Model of Syntax Acquisition In Children), has simulated with great success a number of empirical phenomena in the early acquisition of syntactic categories (Freudenthal et al., 2008, 2009). Our attention has focused on the “optional infinitive” phenomenon and related phenomena, such as the misuse of pronouns. The “optional infinitive” phenomenon concerns typical errors made by children in their use of finite (for example, *goes, went*) and non-finite verb forms (for example, *go, going*). For example, a child would say “her do it” instead of “she does it.” A combination of four features makes the MOSAIC project unique within cognitive science: (a) it uses naturalistic input (utterances spoken by parents interacting with their children in a play setting); (b) it can simulate in detail the pattern and developmental trend of errors; (c) it uses exactly the same model for reproducing a number of empirical phenomena; and (d) it carries out simulations in several languages (so far, English, Dutch, German, French, Spanish; and Q'anjobalan, a Mayan language) with exactly the same model – the only difference

being the maternal input used for training. Compared to other approaches in the field (e.g., connectionism), the extent of MOSAIC's coverage is striking, in particular when one considers that only simple and local learning mechanisms are used. In a nutshell, three interacting factors are essential for explaining MOSAIC's behaviour: rote learning, the creation and use of generative links, and the statistical structure of the input.

General Intelligence: Some Central Questions

Although CHREST does not yet show fully general intelligence, it offers a cognitive architecture whose combined components have proved sufficient to exhibit behaviour that is, in several domains, remarkably close to human behaviour. In this final section, we explore some central questions in artificial intelligence and cognitive science for which, we believe, CHREST provides important insight.

The Role of Concepts

It is generally accepted that acquiring and using concepts is essential for the survival of organisms and their successful interaction with the environment. Without concepts, organisms can show only primitive behaviour, as the lack of generalization means that each new exemplar must be treated separately. A considerable amount of experimental research has been carried out on concept formation and categorization in psychology over the last decades. We know a great deal about how people acquire and utilise concepts, with a number of computational models accounting for different aspects of these empirical data. For example, Gobet et al. (1997) used a model close to CHREST to investigate the role of strategies in concept formation, and the suitability of CHREST for simulating categorization experiments was further established by Lane and Gobet (2005).

The previous simulations with CHREST have essentially relied on individual nodes to account for the concepts used by humans. However, the type of chunking networks created by CHREST suggests other possibilities as well. A natural interpretation of these networks is that concepts do not map into single chunks (or even single templates), but rather correspond to a subset of nodes interlinked, to varying degrees, by lateral links. In this view, concepts are much more distributed than in standard symbolic models, and take on some of the flavour of how concepts are represented in connectionist networks; this idea is expanded upon in Lane, Gobet and Cheng (2000). While this idea is not new, the advantage of using CHREST is to provide mechanisms explaining how nodes and the links between them are acquired autonomously and in real time, how they relate to perceptual input, how they are influenced by the structure of the environment, and how they may integrate information across different modalities.

Embodiment

Although the first experiments with autonomous robots are

fairly old, going back to Grey Walter's (1953) seminal work, it is only in the last twenty years or so that the field of embodied cognition has been taken up. A number of mobile robots have been created that are able to carry out fairly simple tasks (Pfeifer and Scheier, 1999). However, as we have argued elsewhere (Lane and Gobet, 2001), a limitation of current research in this field is that it does not provide mechanisms for how simple behaviours can be linked with more complex behaviours. The limitation is serious, as it is obvious that systems showing general intelligence must have (at least) these two levels of complexity. Specifically, the lack of symbolic processing in current embodied-cognition systems means that there are important limits in the kind of behaviours that can be addressed.

As soon as one tries to link simple behaviours with symbolic processes, the question of symbol grounding arises. The CHREST framework provides a fairly simple explanation for this: as noted above, symbols, that is chunks, are grounded through perception. Importantly, as mentioned above, this idea goes much beyond that of simply linking symbols to external objects, through perception. Chunks also shape how the world is perceived, in two important ways. First, they determine how percepts will be organized. The way former world chess champion Gary Kasparov perceptually groups information of a chess position is different from the way an amateur does. Second, chunks actively direct eye movements and thus determine to which part of the display attention will be heeded.

To show how such an integrated system would work, we are currently working to implement CHREST into a mobile robot. Our first aim is to show that a chunking-based architecture can replicate some of the 'classic' simulations in the literature, such as the avoidance of obstacles and the emergence of complex behaviour. Our belief is that combining symbolic and non-symbolic approaches within an embodied system is likely to have important consequences, both for theory and application. In particular, symbolic information will enable the robot to be 'articulate', explaining, verbally, what and why it takes particular decisions.

Conclusion

We have argued that progress towards understanding general intelligence requires an integrated approach: not just integrating perception with learning, but also integrating high-level and low-level modes of processing. It is unlikely that a satisfactory theory of human intelligence will be developed from a single explanatory framework, so we expect an artificial system with claims to general intelligence to be a unification of diverse approaches.

In this paper, we have set out the case for a symbolic system, which integrates perception with learning. We have shown how the system captures many details of high-level processing in human cognition, and also how it captures physical behaviours, such as details of eye fixations. These successes allow us to propose some ideas of how a complete model of the mind may look.

Central would be a two-way interaction between perception and cognition. This interaction must be coupled with an incremental learning system, capable of acquiring a vast and coherent structure of nodes and links. But also, paradoxically perhaps, the architecture should exhibit strong constraints, such as limited processing time or short-term memory capacities. These limits lie behind some of the key empirical challenges to computational theories of psychological behaviour.

Acknowledgements

This research was supported by the Economics and Social Research Council under grant number RES-000-23-1601.

References

- Anderson, J. R., and Lebière, C. (Eds.). (1998). *The atomic components of thought*. Mahwah, NJ: Erlbaum.
- Chase, W. G., and Simon, H. A. (1973). Perception in chess. *Cognitive Psychology*, 4, 55-81.
- De Groot, A. D., and Gobet, F. (1996). *Perception and memory in chess: Heuristics of the professional eye*. Assen: Van Gorcum.
- Dreyfus, H., and Dreyfus, S. (1986). *Mind over machine*. New York: Free Press.
- Feigenbaum, E. A., and Simon, H. A. (1984). EPAM-like models of recognition and learning. *Cognitive Science*, 8, 305-336.
- Freudenthal, D., Pine, J. M., Aguado-Orea, J. & Gobet, F. (2007). Modelling the developmental patterning of finiteness marking in English, Dutch, German and Spanish using MOSAIC. *Cognitive Science*, 31, 311-341.
- Freudenthal, D., Pine, J. M., & Gobet, F. (2009). Simulating the referential properties of Dutch, German and English Root Infinitives in MOSAIC. *Language Learning and Development*, 5, 1-29.
- Gobet, F. (1997). A pattern-recognition theory of search in expert problem solving. *Thinking and Reasoning*, 3, 291-313.
- Gobet, F. (2009). Using a cognitive architecture for addressing the question of cognitive universals in cross-cultural psychology: The example of awalé. *Journal of Cross-Cultural Psychology*, 40, 627-648.
- Gobet, F., and Jansen, P. (1994). Towards a chess program based on a model of human memory. In H. J. van den Herik, I. S. Herschberg and J. E. Uiterwijk (Eds.), *Advances in Computer Chess 7* (pp. 35-60). Maastricht: University of Limburg Press.
- Gobet, F., Lane, P. C. R., Croker, S., Cheng, P. C.-H., Jones, G., Oliver, I., and Pine, J. M. (2001). Chunking mechanisms in human learning. *Trends in Cognitive Sciences*, 5, 236-243.
- Gobet, F., Richman, H., Staszewski, J., and Simon, H. A. (1997). Goals, representations, and strategies in a concept attainment task: The EPAM model. *The Psychology of Learning and Motivation*, 37, 265-290.
- Gobet, F., and Simon, H. A. (1996). Recall of rapidly presented random chess positions is a function of skill. *Psychonomic Bulletin and Review*, 3, 159-163.
- Gobet, F., and Simon, H. A. (2000). Five seconds or sixty? Presentation time in expert memory. *Cognitive Science*, 24, 651-682.
- Gobet, F., and Waters, A. J. (2003). The role of constraints in expert memory. *Journal of Experimental Psychology: Learning, Memory and Cognition*, 29, 1082-1094.
- Grey Walter, W. *The living brain* (1953). London, UK: Penguin.
- Han, J.W., Lane, P.C.R., Davey, N. and Sun Y. (2009). Attention mechanisms and component-based face detection. *Proceedings of the International Conference on Methods and Models in Computer Science* (IEEE Computer Society).
- Jones, G., Gobet, F., & Pine, J. M. (2007). Linking working memory and long-term memory: A computational model of the learning of new words. *Developmental Science*, 10, 853-873.
- Jones, G., Gobet, F., & Pine, J. M. (2008). Computer simulations of developmental change: The contributions of working memory capacity and long-term knowledge. *Cognitive Science*, 32, 1148-1176.
- Lane, P. C. R., Cheng, P. C.-H., and Gobet, F. (2000). CHREST+: Investigating how humans learn to solve problems using diagrams. *AISB Quarterly*, 103, 24-30.
- Lane, P. C. R., and Gobet, F. (2001). Simple environments fail as illustrations of intelligence: A review of R. Pfeifer and C. Scheier: 'Understanding Intelligence'. *Artificial Intelligence*, 127, 261-267.
- Lane, P. C. R., & Gobet, F. (2005). Discovering predictive variables when evolving cognitive models. *Third International Conference on Advances in Pattern Recognition*.
- Lane, P.C.R., Gobet, F. & Cheng, P.C-H. (2000). Learning-based constraints on schemata. *Proceedings of the Twenty-Second Annual Conference of the Cognitive Science Society*, pp. 776-81.
- Lane, P. C. R., Sykes, A. K., and Gobet, F. (2003). Combining low-level perception with expectations in CHREST. In F. Schmalhofer, R. M. Young and G. Katz (Eds.), *Proceedings of EuroCogSci 03: The European Cognitive Science Conference 2003* (pp. 205-210). Mahwah, NJ: Erlbaum.
- Newell, A. (1990). *Unified theories of cognition*. Cambridge, MA: Harvard University Press.
- Pfeifer, R., and Scheier, C. (1999). *Understanding intelligence*. Cambridge: MIT Press.
- Reber, A. S. (1967). Implicit learning of artificial grammars. *Journal of Verbal Learning and Verbal Behaviour*, 6, 855-863.
- Simon, H. A. (1969). *The sciences of the artificial*. Cambridge, MA: MIT Press.
- Simon, H. A. (1979). *Models of thought* (Vol. 1). New Haven, CT: Yale University Press.
- Simon, H. A., and Gilmarin, K. J. (1973). A simulation of memory for chess positions. *Cognitive Psychology*, 5, 29-46.

A Generic Adaptive Agent Architecture Integrating Cognitive and Affective States and their Interaction

Zulfiqar A. Memon^{1,2}, Jan Treur¹

¹VU University Amsterdam, Department of Artificial Intelligence, De Boelelaan 1081, 1081 HV Amsterdam, The Netherlands

²Sukkur Institute of Business Administration (Sukkur IBA), Air Port Road Sukkur, Sindh, Pakistan

Email: {zamemon, treur}@few.vu.nl URL: <http://www.few.vu.nl/~{zamemon, treur}>

Abstract

In this paper a generic adaptive agent architecture is presented that integrates the interaction between cognitive and affective aspects of mental functioning, based on variants of notions adopted from neurological literature. It is discussed how it addresses a number of issues that have recurred in the recent literature on Cognitive Science and Philosophy of Mind.

Introduction

Recent neurological findings suggest that studying cognitive and affective aspects of mental functioning separately may not be a fruitful way to go. For example, Phelps (2006, pp. 46-47) states: ‘The mechanisms of emotion and cognition appear to be intertwined at all stages of stimulus processing and their distinction can be difficult. (...) Adding the complexity of emotion to the study of cognition can be daunting, but investigations of the neural mechanisms underlying these behaviors can help clarify the structure and mechanisms’. Similar claims have been made recently by Dolan (2002), Pessoa (2008), and others.

This paper describes a generic agent architecture that integrates cognitive and affective aspects of mental functioning and their interaction. Abstracted variants of a number of notions adopted from neurological literature served as ingredients for this architecture: cognitive, affective states, and (causal) relations between such states as having strengths expressed by real numbers, body loops and as if body loops (Damasio, 1999), the mirroring function of preparation neurons (Rizzolatti and Sinigaglia, 2008; Iacoboni, 2008; Pineda, 2009), the process of somatic marking (Damasio, 1994, 2003), and Hebbian learning (Hebb, 1949). In this paper first the generic architecture is described in some detail. Next it is shown how it addresses a number of issues that have been recurring themes in the literature on Cognitive Science and Philosophy of Mind over the last 20 years or more: subjectivity in observing and believing, simulation theory vs theory theory of mind, the reverse causation paradox in mindreading, empathic understanding, adaptivity, rationality and emotion in decision making, causal efficacy of qualia, and physical grounding.

As described in more detail in (Memon and Treur, 2009a; Memon and Treur, 2009b; Bosse, Memon, and

Treur, 2009; Memon and Treur, 2008), the proposed generic agent architecture has been applied by developing specialisations for a number of cases involving, for example, emotion generation, emotion reading, empathic understanding, believing, and trusting. The current paper describes the generic agent architecture behind these specialisations and reviews how it addresses a number of recurring themes in the literature on mind and cognition.

The Generic Adaptive Agent Architecture

A first design choice for the agent architecture was to represent (cognitive and affective) states as having certain levels or extents. This is a crucial choice as it enables dynamics of states by small changes, for example in recursive loops. A related choice is to assign strengths to causal relations between states, which also enables adaptivity. A specific agent model can be designed by using a number of such states and causal relations between them with connection strengths fixed or based on learning. In Figure 2 an overview of such basic elements is given.

Informally described theories in, for example, biological or neurological disciplines, often are formulated in terms of causal relationships or in terms of dynamical systems. To adequately formalise such a theory the hybrid dynamic modelling language LEADSTO has been developed that subsumes qualitative and quantitative causal relationships, and dynamical systems; cf. (Bosse, Jonker, Meij and Treur, 2007). Within LEADSTO the *dynamic property* or temporal relation $a \rightarrow_D b$ denotes that when a state property a occurs, then after a certain time delay (which for each relation instance can be specified as any positive real number D), state property b will occur. Below, this D will be taken as the time step Δt , and usually not be mentioned explicitly. In LEADSTO both logical and numerical calculations can be specified in an integrated manner, and a dedicated software environment is available to support specification and simulation, in which the presented agent model has been formally specified and which was used for simulation experiments.

First the process of sensing is addressed. Here W is a variable for world state properties and V for real values in the interval $[0, 1]$. For an overview, see also Figure 2.

LP1 Sensing a world state

world_state(W, V) &
 connection_strength(world_state(W), sensor_state(W), ω)
 \rightarrow sensor_state(W, $f(\omega V)$)

LP2 Generating a sensory representation for a world state

sensor_state(W, V) &
 connection_strength(sensor_state(W), srs(W), ω)
 \rightarrow srs(W, $f(\omega V)$)

In these specifications, the function $f(W)$ (with $W = \omega V$) can be specified by a threshold function $h(\sigma, \tau, W) = 1/(1 + \exp(-\sigma(W - \tau)))$, with steepness σ , and threshold τ , or simply by the identity function $g(W) = W$.

In principle direct connections can be made from sensory representation states to preparation states (not depicted in Figure 2), or intermediate cognitive states can be used as depicted in Figure 2. Property LP3 describes the response to a cognitive state c in the form of the preparation for a specific (bodily) reaction b . This specifies part of the recursive loop between cognitive and affective states; see Figure 1. It is calculated based on a parameterised function $f(W_1, W_2)$ of the original levels V_i (with $W_i = \omega_i V_i$).

LP3 Generating a preparation state

cognitive_state(c, V_1) & feeling(b, V_2) & preparation_state(b, V_3)
 & connection_strength(cognitive_state(c), preparation_state(b), ω_1)
 & connection_strength(feeling(b), preparation_state(b), ω_2)
 \rightarrow preparation_state(b, $V_3 + \gamma(f(\omega_1 V_1, \omega_2 V_2) - V_3) \Delta t$)

In different applications of the generic agent architecture, two templates of functions $f(W_1, W_2)$ have been taken:

$$g(\beta, W_1, W_2) = \beta(1 - (1 - W_1)(1 - W_2)) + (1 - \beta)W_1W_2$$

$$h(\sigma, \tau, W_1, W_2) = 1/(1 + \exp(-\sigma(W_1 + W_2 - \tau)))$$

Note that the latter formula is often used as threshold function in neural models. The first formula describes a weighted sum of two cases. The most positive case considers the two source values as strengthening each other, thereby staying under 1: combining the imperfection rates $1 - W_1$ and $1 - W_2$ of them provides a decreased rate of imperfection $1 - (1 - W_1)(1 - W_2)$. The most negative case considers the two source values in a negative combination: combining the imperfections of them provides an increased imperfection expressed by W_1W_2 . The factor β is a characteristic that expresses the person's orientation (from 0 as most negative to 1 as most positive). The parameter γ indicates the speed of change: how flexible the state is.

A further choice was to use body loops and as if body loops for preparations of actions, adopted from (Damasio, 1999, 2003; Bosse, Jonker, and Treur, 2008). This provides a second type of recursive loop: between preparation and feeling states (see Figure 1).

Thus a combination of two loops is obtained, where connection strengths within these loops in principle are person-specific (and might be subject to learning). Depending on these personal characteristics, from a dynamic interaction within and between the two loops, for

a given stimulus an equilibrium is reached for the strength of the cognitive, preparation, and feeling state.

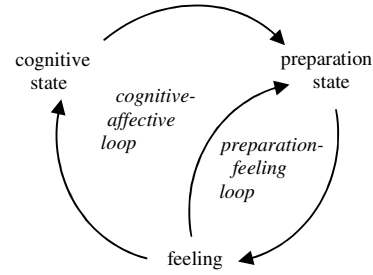


Figure 1: The two recursive loops related to a cognitive state

The existence of a connection from feeling to cognitive state can be supported by Damasio's Somatic Marker Hypothesis; cf. (Damasio, 1994, 1996, 2003; Bechara and Damasio, 2004). This is a theory on decision making which provides a central role to emotions felt. Each decision option induces (via an emotional response) a feeling which is used to mark the option. For example, when a negative somatic marker is linked to a particular option, it provides a negative feeling for that option. Similarly, a positive somatic marker provides a positive feeling for that option. Usually the Somatic Marker Hypothesis is applied to provide endorsements or valuations for options for a person's actions. However, it may be considered plausible that such a mechanism is applicable to valuations of internal cognitive states (e.g., beliefs) as well. In detail the recursive loops between preparation and feeling are specified by LP4 to LP8 (see also Figure 2). Here B is a variable for body states.

LP4 From preparation to effector state for body modification

preparation_state(B, V) &
 connection_strength(preparation_state(B), effector_state(B), ω)
 \rightarrow effector_state(B, $f(\omega V)$)

LP5 From effector state to modified body state

effector_state(B, V) &
 connection_strength(effector_state(B), body_state(B), ω)
 \rightarrow body_state(B, $f(\omega V)$)

LP6 Sensing a body state

body_state(B, V) &
 connection_strength(body_state(B), sensor_state(B), ω)
 \rightarrow sensor_state(B, $f(\omega V)$)

LP7 Generating a sensory representation of a body state

sensor_state(B, V_1) & preparation_state(B, V_2) & srs(B, V_3) &
 connection_strength(preparation_state(B), srs(B), ω_1) &
 connection_strength(sensor_state(B), srs(B), ω_2)
 \rightarrow srs(B, $V_3 + \gamma(f(\omega_1 V_1, \omega_2 V_2) - V_3) \Delta t$)

LP8 From sensory representation of body state to feeling

srs(B, V) & connection_strength(srs(B), feeling(B), ω)
 \rightarrow feeling(B, $f(\omega V)$)

Next the property for generation of the cognitive state c is described, where both a sensory representation of w and a feeling of b play their role. This specifies another part of

the loop between cognitive and affective states, in addition to LP3 (see Figure 1). The resulting level for the cognitive state c is calculated based on a parameterised function $f(U_1, U_2)$ of the original levels V_i (with $U_i = \omega_i V_i$). Here w , c , and b are specific instances of world state, cognitive state and body state.

LP9 Generating a cognitive state

$\text{srs}(w, V_1) \ \& \ \text{feeling}(b, V_2) \ \& \ \text{cognitive_state}(c, V_3) \ \& \ \text{connection_strength}(\text{srs}(w), \text{cognitive_state}(c), \omega_1) \ \& \ \text{connection_strength}(\text{feeling}(b), \text{cognitive_state}(c), \omega_2)$
 $\rightarrow \text{cognitive_state}(c, V_3 + \gamma_3 (f(\omega_1 V_1, \omega_2 V_2) - V_3) \Delta t)$

Within the agent architecture the adaptive element is incorporated by making (some of the) connection strengths adaptive. From a Hebbian neurological perspective (Hebb, 1949), strengthening of connections over time may be considered plausible, when neurons involved in the connected nodes often are activated simultaneously. Therefore such a connection can be adapted based on a Hebbian learning mechanism (Hebb, 1949; Bi and Poo, 2001; Gerstner and Kistler, 2002). Based on these considerations, in the agent architecture connection strengths ω can be adapted using the following Hebbian learning rule. It takes into account a maximal connection strength l , a learning rate η , and an extinction rate ζ . A similar Hebbian learning rule can be found in (Gerstner and Kistler, 2002, p. 406). Here N_i are variables over neural states.

LP10 Hebbian learning rule template

$N_1(V_1) \ \& \ N_2(V_2) \ \& \ \text{connection_strength}(N_1, N_2, \omega) \ \& \ \text{learning_rate}(N_1, N_2, \eta) \ \& \ \text{extinction_rate}(N_1, N_2, \zeta)$
 $\rightarrow \text{connection_strength}(N_1, N_2, \omega + (\eta V_1 V_2 (1 - \omega) - \zeta \omega) \Delta t)$

By the factor $1 - \omega$ the learning rule keeps the level of ω bounded by l (which could be replaced by any number). When extinction is neglected, the upward changes during learning are proportional to both V_1 and V_2 , which in particular means that no learning takes place whenever one of them is 0 , and maximal learning takes place when both are l .

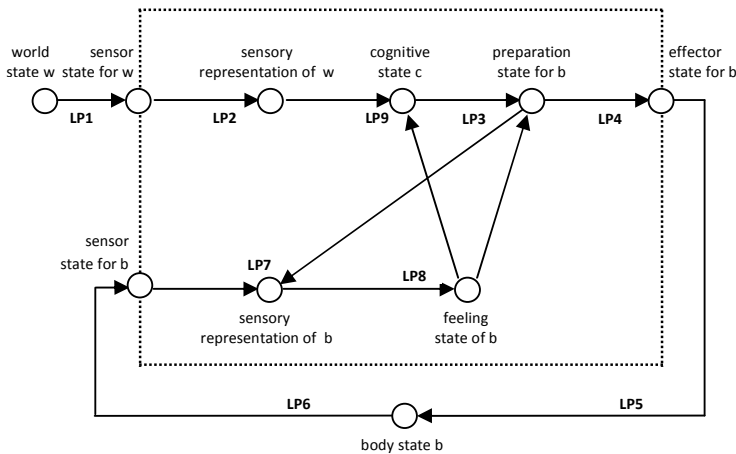


Figure 2: Basic elements of the architecture

How the Different Issues Are Addressed

This section addresses how the presented agent architecture addresses a number of recurring issues in the literature on cognition and mind.

• The role of subjectivity in observing and believing

How states and characteristics of the subject affect his or her observation and beliefs is a first theme discussed. In an idealised rational agent the generation of cognitive states might only depend on informational sources and be fully independent from non-informational aspects such as emotions. However, in real life persons may, for example, have a more optimistic or pessimistic character and affect their beliefs and other cognitive states in the sense that an optimist person strengthens beliefs that have a positive feeling associated and a pessimistic person strengthens beliefs with a negative associated feeling. Thus the strengths may depend on non-informational aspects of mental processes and related personal characteristics. The presented agent architecture allows to associate emotional responses to sensory representations that are felt (through connections LP3, LP4, LP5, LP6, LP7, LP8), and affect further cognitive processing (through LP9). This has been worked out in more detail for beliefs in (Memon and Treur, 2009a). For this specialisation in Figure 2 for the cognitive state c , the belief that w holds is taken.

• The reverse causation paradox in mindreading

A next issue discussed is in how far mindreading makes use of the own mental states that are counterparts of the attributed mental states. A nontrivial obstacle for the Simulation Theory perspective (e.g., Goldman, 2006) on mindreading is what can be called the *reverse causation paradox*: how to simulate a process which in principle has the reverse order compared to the causal relations used in the simulation. Further analysis reveals that this paradox in fact originates from the often made assumption that the causal relations used by the observed person flow from mental states to actions and body states, whereas the latter is what the observing agent observes. As within the observing agent this observation is the starting point for the simulation process to determine the observed person's mental states, this would be against the direction of the causal relations used. This issue is encountered, for example, in (Goldman and Sripada, 2004; Goldman, 2006, pp. 124-132), where four possible informal emotion reading models from the Simulation Theory perspective are sketched and discussed. Partly inspired by these models, in (Memon and Treur, 2008) an emotion reading model was introduced, based on the agent architecture presented here, to address this paradox.

In (Goldman, 2006), for model 1, to resolve the reverse causation paradox, a 'generate and test process' for emotional states was assumed, where on the basis of a hypothesized emotional state an own facial expression is generated, and this is compared to the observed facial expression of the other person. In the assessment of this model, the unspecified hypothesis generation process for a

given observed face was considered as a less satisfactory aspect. Models 2 and 3 discussed in (Goldman, 2006) are based on a notion of what he calls ‘reverse simulation’. This means that for the causal relation from emotional state to (the preparation of) a facial expression which is used to generate the own facial expressions, also a reverse relation from prepared own facial expression to emotional state is assumed, which is used for the mind reading process. A point of discussion concerning these two models is that whereas the emotional states and facial expression (preparation) states used for mindreading are the same as used for the own emotions and facial expressions, the causal relations between them used in the two cases are not the same. Model 4 is based on a so-called ‘mirroring process’, where a correlation between the emotional state of the other person and the corresponding own emotional state is assumed, based on a certain causal chain between the two. However, the relation of such a causal chain with the causal relations used to generate the own emotional states and facial expressions is not made clear.

The model presented in (Memon and Treur, 2008) based on the agent architecture presented in the current paper, addresses the reverse causation paradox in the following manner. The picture for this specialisation is as in Figure 2, but with cognitive state *c* left out: the sensory representation of *w* has a direct connection to the preparation for *b* (which is assumed to have a mirroring function). Moreover, an additional cognitive imputation state is present to connect the own feeling and the stimulus representing the other person’s face. Within this specialised model the recursive (as if) body loop addresses the problems of Goldman’s model 1, as it can be viewed as an efficient and converging way of generating and testing hypotheses for the emotional states, where the (as if) body loop takes care of the generation process. Moreover, it solves the problems of models 2 and 3, as the causal chain used from facial expression to emotional state is not a reverse simulation, but just the circular causal chain (recursive body loop) which is used for generating the own responses and feeling states as well. Finally, compared to model 4, the model put forward in (Memon and Treur, 2008) can be viewed as an efficient manner to obtain a mirroring process between the emotional state of the other person on the own emotional state, based on the machinery available for the own emotional states.

- *Empathic understanding*

For humans, one of the deepest and most fundamental forms of mutual understanding is based on the notion of *empathy*; e.g., (Ickes, 1997; Preston and Waal, 2002; Decety and Jackson, 2004; Lamm, Batson, and Decety, 2007; Iacoboni, 2005, 2008). Originally (cf. Lipps, 1903) the notion of empathy was named by the German word ‘*einfühlung*’ which could be translated as ‘feeling into’; e.g., (Preston and Waal, 2002). As this word indicates more explicitly, the notion has a strong relation to feeling: *empathic understanding* is a form of understanding which includes (but is not limited to) feeling what the other person feels. A particular challenge here is how to enrich

understanding of any cognitive state (such as an attention, belief, desire or intention state) of another person to a form of understanding which includes feeling the same emotion associated to this cognitive state as the other person. So a crucial aspect for empathic understanding is the way in which feelings and other mental states are interrelated. For example, a belief that something bad is to happen, may relate to feeling fear, or the belief that something good has happened may relate to feeling happiness. Another example of such a relationship is the role of cognitive elements (for example, certain thoughts) in the development, persistence and recurrence of mood disorders such as depressions; e.g., (Ingram, Miranda & Segal, 1998). So, in empathic understanding both cognitive and affective states are to be involved in their mutual relationship, of both the observed and observing person.

In (Memon and Treur, 2009b) this challenging notion of empathic understanding was addressed, based on the agent architecture presented here. The model describes how the empathic agent does not only understand another agent’s cognitive state but at the same time feels the accompanying emotion. It was based on two main assumptions:

- (1) The observing agent performs mindreading using the same mental states as the observed agent
- (2) Both agents have a similar mechanism to associate feelings to a given cognitive state

Concerning assumption (1), the Simulation Theory perspective was followed; cf. (Goldman, 2006). For assumption (2) the body loop was exploited. Assuming that the observed agent and the observing agent indeed have a similar mechanism for this, makes it possible that for a given cognitive state the observing agent generates the same feeling as the observed agent. The picture for this specialisation is as in Figure 2, with *c* the cognitive state for which empathic understanding takes place. Moreover, two additional cognitive imputation states are present to connect both the cognitive state *c* and the own feeling state to the other person.

Especially in relation to assumption (2) it can be questioned to which extent the mechanisms to associate feelings to a given mental state are always the same for two persons. As it may be considered plausible that basically the mechanisms are similar, it is not difficult to imagine that both due to innate and learned individual differences in the strengths of the connections in the body loops, the extent of the empathic reaction may differ. Indeed, it is often reported that identical twins have a much higher level of mutual empathy than any two persons which are not identical twins. Moreover, it is also often considered that more empathy is shown between two persons when they have had similar experiences in life. Nevertheless, a certain extent of empathy still seems possible between persons which are not genetically identical and have not exactly the same experiences. The connection strengths may be considered parameters by which such innate and acquired individual differences can be characterised.

- *Adaptivity*

The issue of adaptivity concerns how a transparent mechanism can be obtained describing how the internal mental processes adapt to the subject's experiences with the world. Classical symbolic models usually have limited or no adaptivity, as states are often taken binary. In the agent architecture presented here such adaptivity is modelled based on the fact that states and connections are expressed by real numbers, and by use of an abstracted variant of Hebb's learning principle (Hebb, 1949; Bi and Poo, 2001; Gerstner and Kistler, 2002). It has been shown in a number of cases how this enables an agent to adapt to experiences with the world, for example, involving notions such as trust based on experiences, and learning direct emotion recognition by a form of classification of face expressions, (Bosse, Memon, and Treur, 2009; Hoogendoorn, Jaffry, and Treur, 2009; Jaffry and Treur, 2009). In (Memon, Treur, and Umair, 2009) it has been analysed how this Hebbian learning principle compares to some other well-known learning principles based on temporal discounting and memory traces.

- *Rationality and emotion in decision making*

Another recurring theme is how rational and emotional aspects in decision making can be understood in a coherent fashion. A neurological theory addressing the interaction between cognitive and affective aspects in decision making is Damasio's Somatic Marker Hypothesis; cf. (Damasio, 1994, 1996; Bechara and Damasio, 2004; Damasio, 2003). This is a theory on decision making which provides a central role to emotions felt. Within a given context, each represented decision option induces (via an emotional response) a feeling which is used to mark the option. For example, a strongly negative somatic marker linked to a particular option occurs as a strongly negative feeling for that option. Similarly, a positive somatic marker occurs as a positive feeling for that option. This theory provides an account on how emotions and rational aspects cooperate in the decision process and also explains how this can take the form of an adaptive process leading to decision making by intuition or based on 'experience'. This has been exploited in the context of the role of trust states in decision making, modelled based on the agent architecture presented here, in (Hoogendoorn, Jaffry, and Treur, 2009; Jaffry and Treur, 2009).

- *Physical grounding of agent models*

Yet another recurring theme is in how far agent models are embodied, or have some form of physical grounding. Agent models can be designed at different levels of abstraction. For example, the well-known BDI-model makes use of higher-level cognitive concepts such as beliefs, desires and intentions. In order to ground models for embodied agents in a physical, chemical or neurological context, often the focus is on their interaction as a coupled system with the environment; e.g., (Clancey, 1997; Clark, 1997). However, they can be related to physical reality in a still more fundamental manner when the model of their internal functioning is fully immersed in

a model of the world's dynamics, and to this end concepts from a lower level are used in the model, or it is indicated how the concepts used in the model relate to such lower-level concepts. In this way cognition can be addressed by an artificial life like approach; e.g., (Steels and Brooks, 1995; Port and van Gelder, 1995). The agent architecture presented has adopted abstracted variants of a number of neurological principles, among which body loops, Hebbian learning, and a mirroring function of preparation states. Therefore it is easily embeddable in a model at the physiological and neurological level. Such an embedding has been described in more detail in (Memon and Treur, 2008; Treur, 2010).

- *Causal efficacy of feelings and qualia*

A recurring theme in relation to conscious experiencing or qualia is whether qualia have causal efficacy; e.g., (Kim, 1996, pp. 155-183; Duch, 2005). In neurological literature such as (Damasio 1999), qualia have been associated to certain representations of body states. Within the presented agent architecture specific types of body states and feelings associated to internal representations of them, take part in recursive loops. As such these feeling states can be said to be both causing and caused by preparation states and cognitive states. Their efficacy with respect to the preparation and cognitive states through these loops depends on the strengths of the connections in the loops. This shows that at least such feeling states have a certain extent of causal efficacy. Of course, these feeling states may be considered as being rather simplified as compared to qualia, so it is still open for discussion in how far this pattern can also count as a perspective on the question of causal efficacy of qualia.

Discussion

The generic agent architecture presented in this paper integrates the interaction between cognitive and affective aspects of mental functioning, using abstracted variants of notions adopted from neurological literature: cognitive, affective states, and (causal) relations between such states with strengths expressed by real numbers, body loops and as if body loops (Damasio, 1999), the mirroring function of preparation neurons (Rizzolatti and Sinigaglia, 2008; Iacoboni, 2008; Pineda, 2009), the process of somatic marking (Damasio, 1994, 2003), and Hebbian learning (Hebb, 1949).

This agent architecture has been applied by developing specialisations for a number of cases involving, for example, emotion generation, emotion reading, empathic understanding, believing, and trusting. The current paper describes the generic agent architecture behind these specialisations and reviews how it addresses a number of recurring themes in the literature on mind and cognition.

The architecture has been formally specified in the hybrid modelling language LEADSTO (Bosse et al., 2007). It allows for both mathematical analysis and logical

analysis (verification) as has been shown in work on specialisations of the architecture, such as (Bosse, Memon and Treur, 2009; Memon and Treur, 2009a, 2009b).

The generic agent architecture illustrates how recent developments within the neurological area can be adopted and shaped to obtain innovative design elements for agent models. It shows how cognitive modelling and artificial (general) intelligence can benefit of such developments.

References

- Bechara, A., and Damasio, A. (2004). The Somatic Marker Hypothesis: a neural theory of economic decision. *Games and Economic Behavior*, vol. 52, pp. 336-372.
- Bi, G.Q., and Poo, M.M. (2001) Synaptic Modifications by Correlated Activity: Hebb's Postulate Revisited. *Ann Rev Neurosci*, vol. 24, pp. 139-166.
- Bosse, T., Jonker, C.M., Meij, L. van der, and Treur, J., (2007). A Language and Environment for Analysis of Dynamics by Simulation. *International Journal of Artificial Intelligence Tools*, vol. 16, 2007, pp. 435-464.
- Bosse, T., Jonker, C.M., and Treur, J., (2008). Formalisation of Damasio's Theory of Emotion, Feeling and Core Consciousness. *Consciousness and Cognition Journal*, vol. 17, 2008, pp. 94-113.
- Bosse, T., Memon, Z.A., and Treur, J., (2009). An Adaptive Agent Model for Emotion Reading by Mirroring Body States and Hebbian Learning. In: Yang, J.-J., et al. (eds.), *Proc. of the 12th International Conference on Principles of Practice in Multi-Agent Systems, PRIMA'09*. Lecture Notes in Artificial Intelligence, vol. 5925, Springer Verlag, 2009, pp. 552-562.
- Clancey, W. (1997). *Situated Cognition: On Human Knowledge and Computer Representations*. Cambridge University Press.
- Clark, A. (1997). *Being There: Putting Brain, Body and World Together Again*. Cambridge, MA: MIT Press.
- Damasio, A. (1994). *Descartes' Error: Emotion, Reason and the Human Brain*, Papermac, London.
- Damasio, A. (1996). The Somatic Marker Hypothesis and the Possible Functions of the Prefrontal Cortex. *Philosophical Transactions of the Royal Society: Biological Sciences*, vol. 351, pp. 1413-1420.
- Damasio, A. (1999). *The Feeling of What Happens. Body and Emotion in the Making of Consciousness*. New York: Harcourt Brace, 1999.
- Damasio, A. (2003). *Looking for Spinoza: Joy, Sorrow, and the Feeling Brain*. Vintage books, London, 2004.
- Decety, J. and Jackson, P.L. (2004) The functional architecture of human empathy. *Behav. Cogn. Neurosci. Rev.* 3, 71-100.
- Dolan, R.J. (2002). Emotion, Cognition, and Behavior. *Science*, vol 298, 2002, pp. 1191-1194.
- Gerstner, W., and Kistler, W.M. (2002). Mathematical formulations of Hebbian learning. *Biol. Cybern.*, vol. 87, 2002, pp. 404-415.
- Goldman, A.I. (2006). *Simulating Minds: The Philosophy, Psychology, and Neuroscience of Mindreading*. New York: Oxford Univ. Press.
- Goldman, A.I., and Sripada, C.S. (2004). Simulationist models of face-based emotion recognition. *Cognition*, vol. 94, pp. 193-213.
- Hebb, D.O. (1949). *The Organization of Behaviour*. John Wiley & Sons, New York, 1949.
- Hoogendoorn, M., Jaffry, S.W., and Treur, J., (2009). Modelling Trust Dynamics from a Neurological Perspective. In: *Proc. of the Second International Conference on Cognitive Neurodynamics, ICCN'09*. Springer Verlag, 2009, to appear.
- Iacoboni M. (2008). *Mirroring People: the New Science of How We Connect with Others*. New York: Farrar, Straus & Giroux
- Ickes, W. (1997). *Empathic Accuracy*. Guilford Press, New York.
- Jaffry, S.W., and Treur, J., (2009). Comparing a Cognitive and a Neural Model for Relative Trust Dynamics. In: *Proceedings of the 16th International Conference on Neural Information Processing, ICONIP'09*. Lecture Notes in Computer Science, vol. 5863. Springer Verlag, 2009, pp. 72-83
- Kim, J. (1996). *Philosophy of Mind*. Westview Press.
- Lamm, C., Batson, C.D., and Decety, J. (2007). The neural basis of human empathy – effects of perspective-taking and cognitive appraisal. *J. Cogn. Neurosci.*, vol. 19, 2007, pp. 42-58.
- Lipps, T. (1903) Einfühlung, innere Nachahmung und Organempfindung. *Archiv für die gesamte Psychologie*, vol. 1, pp. 465-519.
- Memon, Z.A., and Treur, J., (2008). Cognitive and Biological Agent Models for Emotion Reading. In: Jain, L., et al. (eds.), *Proceedings of the 8th IEEE/WIC/ACM International Conference on Intelligent Agent Technology, IAT'08*. IEEE Computer Society Press, 2008, pp. 308-313.
- Memon, Z.A., and Treur, J., (2009a). Modelling the Reciprocal Interaction between Believing and Feeling from a Neurological Perspective. In: N. Zhong et al. (eds.), *Proc. of the First Intern. Conf. on Brain Informatics, BI'09*. Lecture Notes in Artificial Intelligence, vol. 5819. Springer Verlag, 2009, pp. 13-24.
- Memon, Z.A., and Treur, J., (2009b). Designing Social Agents with Empathic Understanding. In: Nguyen, N.T., et al. (eds.), *Proc. of the First International Conference on Computational Collective Intelligence, ICCCI'09*. Lecture Notes in Artificial Intelligence, vol. 5796. Springer Verlag, 2009, pp. 279-293.
- Memon, Z.A., Treur, J., and Umair, M., (2009). A Comparative Analysis on Adaptive Modelling of Induced Feelings. In: *Proc. of the Second International Conference on Cognitive Neurodynamics, ICCN'09*. Springer Verlag, 2009, to appear.
- Pessoa, L. (2008). On the relationship between emotion and cognition. *Nature Reviews: Neuroscience*, vol. 9, 2008, pp. 148-158.
- Phelps, E.A. (2006). Emotion And Cognition: Insights from Studies of the Human Amygdala. *Annu. Rev. Psychol.* 2006. 57:27-53
- Pineda, J.A. (ed.), (2009). *Mirror Neuron Systems: the Role of Mirroring Processes in Social Cognition*. Humana Press Inc.
- Port, R.F., Gelder, T. van (eds.), (1995). *Mind as Motion: Explorations in the Dynamics of Cognition*. MIT Press, Cambridge, Mass, 1995.
- Preston, S.D. and Waal, F.B.M. de (2002). Empathy: its ultimate and proximate bases. *Behav. Brain Sci.* 25, 1-72.
- Rizzolatti, G. and Sinigaglia, C., (2008). *Mirrors in the Brain: How Our Minds Share Actions and Emotions*. Oxford University Press, 2008.
- Steels, L. & Brooks, R. (1995). *The artificial life route to artificial intelligence: Building embodied, situated agents*. Erlbaum.
- Duch, W. (2005), Brain-inspired conscious computing architecture. *Journal of Mind and Behavior* 26(1-2) (2005) 1-22
- Treur, J., (2010). On the Use of Reduction Relations to Relate Different Types of Agent Models. *Web Intelligence and Agent Systems Journal*, 2010, to appear.

Algorithmic Probability, Heuristic Programming and AGI

Ray J. Solomonoff

Visiting Professor, Computer Learning Research Center
Royal Holloway, University of London

IDSIA, Galleria 2, CH-6928 Manno-Lugano, Switzerland
rjsolo@ieee.org <http://world.std.com/~rjs/pubs.html>

Introduction

This paper is about Algorithmic Probability (ALP) and Heuristic Programming and how they can be combined to achieve AGI. It is an update of a 2003 report describing a system of this kind (Sol03). We first describe ALP, giving the most common implementation of it, then the features of ALP relevant to its application to AGI.

They are: Completeness, Incomputability, Subjectivity and Diversity. We then show how these features enable us to create a very general, very intelligent problem solving machine. For this we will devise “Training Sequences” — sequences of problems designed to put problem-solving information into the machine. We describe a few kinds of training sequences.

The problems are solved by a “generate and test” algorithm, in which the candidate solutions are selected through a “Guiding Probability Distribution”. The use of Levin’s search procedure enables us to efficiently consider the full class of partial recursive functions as possible solutions to our problems. The guiding probability distribution is updated after each problem is solved, so that the next problem can profit from things learned in the previously solved problems.

We describe a few updating techniques. Improvements in updating based on heuristic programming is one of the principal directions of current research. Designing training sequences is another important direction.

For some of the simpler updating algorithms, it is easy to “merge” the guiding probabilities of machines that have been educated using different training sequences — resulting in a machine that is more intelligent than any of the component machines.

What is Algorithmic Probability?

ALP is a technique for the extrapolation of a sequence of binary symbols — all induction problems can be put into this form. We first assign a probability to any finite binary sequence. We can then use Bayes’ theorem to compute the probability of any particular continuation sequence. The big problem is: how do we assign these probabilities to strings? In one of the commonest implementations of ALP, we have a Universal Turing

Machine with three tapes: a unidirectional input tape, a unidirectional output tape, and a bidirectional work tape. If we feed it an input tape with 0’s and 1’s on it, the machine may print some 0’s and 1’s on the output — it could print nothing at all or print a finite string and stop or it could print an infinite output string, or it could go into an infinite computing loop with no printing at all.

Suppose we want to find the ALP of finite string x . We feed random bits into the machine. There is a certain probability that the output will be a string that starts out with the string x . *That* is the ALP of string x .

To compute the ALP of string x :

$$P_M(x) = \sum_{i=0}^{\infty} 2^{-|S_i(x)|}$$

Here $P_M(x)$ is the ALP (also called Universal Probability) of string x with respect to machine, M .

There are many finite string inputs to M that will give an output that begins with x . We call such strings “codes for x ”. Most of these codes are redundant in the sense that if one removes its most recent bit the resultant string will still be a “code for x ”. A “minimal code for x ” is one that is not redundant. If one removes its last bit, the result will no longer be a “code for x ”. Say $|S_i(x)|$ is the length in bits of the i^{th} “Minimal code for x ”.

$2^{-|S_i(x)|}$ is the probability that the random input will begin with the “ i^{th} minimal code for x ”.

$P_M(x)$ is then the sum of the probabilities of all the ways that a string beginning with x , could be generated.

This definition has some interesting properties:

First, it assigns high probabilities to strings with short descriptions — This is in the spirit of Ockham’s razor. It is the converse of Huffman coding that assigns short codes to high probability symbols.

Second, its value is somewhat independent of what universal machine is used, because codes for one universal machine can always be obtained from another universal machine by the addition of a finite sequence of translation instructions.

A less apparent but clearly desirable property — $P_M(x)$ is *complete*. This means that if there is any

describable regularity in a batch of data, P_M will find it, using a relatively small amount of the data. At this time, it is the only induction method known to be complete (Sol78).

More exactly: Suppose $\mu(x)$ is a probability distribution on finite binary strings. For each $x = x_1, x_2 \dots x_i$, μ gives a probability that the next bit, x_{i+1} will be 1: $\mu(x_{i+1} = 1 | x_1, x_2 \dots x_i)$

From P_M we can obtain a similar function $P(x_{i+1} = 1 | x_1, x_2 \dots x_i)$.

Suppose we use μ to generate a sequence, x , Monte Carlo-wise. μ will assign a probability to the $i + 1^{th}$ bit based on all previous bits. Similarly, P will assign a probability to the $i + 1^{th}$ bit of x . If P_M is a very good predictor, the probability values obtained from μ and from P_M will be very close, on the average, for long sequences. What I proved was:

$$E_{\mu} \sum_{i=1}^n (\mu(x_{i+1} = 1 | x_1, x_2 \dots x_i) - P(x_{i+1} = 1 | x_1, x_2 \dots x_i))^2 \leq \frac{1}{2} k \ln 2$$

The expected value of the sum of the squares of the differences between the probabilities is bounded by about $.35k$. k is the minimum number of bits that M , the reference machine, needs to describe μ . If the function μ is describable by functions that are close to M 's primitive instruction set, then k will be small and the error will be small. But whether large or small, the squared error in probability must converge faster than $\frac{1}{n}$ (because $\sum \frac{1}{n}$ diverges).

Later research has shown this result to be very robust — we can use a large, (non-binary) alphabet and/or use error functions that are different from total square difference (Hut02). The probability obtained can be normalized or unnormalized (semi-measure)(Gác97).

The function μ to be “discovered” can be any describable function — primitive recursive, total recursive, or partial recursive. When ALP uses an unnormalized semi-measure, it can discover *incomputable* functions as well.

The desirable aspects of ALP are quite clear. We know of no other model of induction that is nearly as good.

An apparent difficulty — $P_M(x)$ is *incomputable*: The equation defining $P_M(x)$ tells us to find all strings that are “minimal codes for x .” Because of the Halting Problem, it is impossible to tell whether certain strings are codes for x or not. While it is easy to make approximations to $P_M(x)$, the fact that it is incomputable has given rise to the common misconception that ALP is little more than an interesting theoretical model with no direct practical application. We will show that *In Fact* incomputability is a *desirable* feature and imposes no serious restrictions on its application to the practical problems of AGI.

The usual question is — “What good is it if you can’t compute it?” The answer is that for practical prediction we don’t have to compute ALP *exactly*. Approximations to it are quite usable and *the closer an approximation is to ALP, the more likely it is to share ALP’s desirable qualities*.

Perhaps the simplest kind of approximation to an incomputable number involves making rational approximations to $\sqrt{2}$. We know that there is no rational number whose square is 2, but we can get arbitrarily close approximations. We can also compute an upper bound on the error of our approximation and for most methods of successive approximation we are assured that the errors approach zero. In the case of ALP, though we are assured that the approximations will approach ALP arbitrarily closely, the incomputability implies that we *cannot ever* compute useful upper bounds on approximation error — but *for few if any practical applications do we need this information*.

The approximation problem for the universal distribution is very similar to that of approximating a solution to the Traveling Salesman Problem, when the number of cities is too large to enable an exact solution. When we make trial paths, we always know the total length of each path — so we know whether one trial is better than another. In approximations for the universal distribution, we also always know when one approximation is better than another — and we know how much better. In some cases, we can combine trials to obtain a trial that is better than either of the component trials. In both TSP and ALP approximation, we never know how far we are from the theoretically best, yet in both cases we do not hesitate to use approximate solutions to our problems.

The incomputability of ALP is closely associated with its completeness. Any complete induction system cannot be computable. Conversely, any computable induction system cannot be complete. For any computable induction system, it is possible to construct a space of data sequences for which that system gives *extremely* poor probability values. The sum of the squared errors diverges linearly in the sequence length.

Appendix B gives a simple construction of this kind.

We note that the incomputability of ALP makes such a construction impossible and its probability error always converges to zero for *any* finitely describable sequence.

To explain our earlier remark on incomputability as a very *desirable* feature: Incomputability is the *only* way we can achieve completeness. In ALP this incomputability imposes no penalty on its practical application. It is a true “Win, Win” situation!

Another item of importance: For most applications an estimate of future prediction error is needed. Cross Validation or one of its many variants is usually possible. In this aspect of the prediction problem ALP is certainly no worse than any other method. On the other hand, ALP gives a good theoretical framework that enables us to make better estimates.

Subjectivity

Occasionally in making extrapolations of a batch of data, there is enough known about the data so that it is clear that a certain prediction technique is optimal. However, this is often not the case and the investigator must make a (subjective) choice of inductive techniques. For me, the choice is clear: I choose ALP because it is the only complete induction method I know of. However ALP has another subjective aspect as well: we have to choose M , the reference machine or language. As long as M is universal, the system will be complete. This choice of M enables us to insert into the system any a priori information we may have about the data to be predicted and still retain completeness.

The choice of the M can be used very effectively for incremental learning: Suppose we have 100 induction problems: X_1, X_2, \dots, X_{100} .

The best solution would involve getting the machine to find a short code for the entire batch of 100 problems. For a large corpus this can be a lengthy task. A shorter, approximate way: using the machine M as reference, we find a prediction code for X_1 . In view of this code, we modify M to become M' in such a way that if $P_{M'}$ makes a prediction for X_2 , it will be the same as if we used P_M for both X_1 and X_2 . M' becomes a *complete summary* of M s increase in knowledge after solving X_1 . We can consider the M to M' transformation as an *updating* of the M to M' . It is possible to show that such an M can be found *exactly*, but the exact construction is very time consuming (reference). Approximations to M' can be readily found. After M' solves X_2 , M' can be updated to M'' and have it solve X_3 , and so on to X_{100} . We will discuss the update process later in more detail, for a somewhat different kind of problem.

To understand the role of subjectivity in the life of a human or an intelligent machine, let us consider the human infant. It is born with certain capabilities that assume certain a priori characteristics of its environment-to-be. It expects to breathe air, its immune system is designed for certain kinds of challenges, it is usually able to learn to walk and converse in whatever human language it finds in its early environment. As it matures, its a priori information is modified and augmented by its experience.

The inductive system we are working on is of this sort. Each time it solves a problem or is unsuccessful in solving a problem, it updates the part of its a priori information that is relevant to problem solving techniques. In a manner very similar to that of a maturing human being, its a priori information grows as the life experience of the system grows.

From the foregoing, it is clear that the subjectivity of algorithmic probability is a necessary feature that enables an intelligent system to incorporate experience of the past into techniques for solving problems of the future.

Diversity

In Section 1 we described ALP based on a universal Turing machine with random input. An equivalent model considers all prediction methods, and makes a prediction based on the weighted sum of all of these predictors. The weight of each predictor is the product of two factors: the first is the a priori probability of each predictor. It is the probability that this predictor would be described by a universal Turing machine with random input. If the predictor is described by a small number of bits, it will be given high a priori probability. The second factor is the probability assigned by the predictor to the data of the past that is being used for prediction. We may regard each prediction method as a kind of model or explanation of the data. Many people would use only the best model or explanation and throw away the rest. Minimum Description Length (Ris78), and Minimum Message Length (WB68) are two commonly used approximations to ALP that use only the best model of this sort. When one model is much better than any of the others, then Minimum Description Length and Minimum Message Length and ALP give about the same predictions. If many of the top models have about the same weight, then ALP gives better results — the other methods give too much confidence in the predictions of the very best model (PH06).

However, that's not the *main* advantage of ALP's use of a diversity of explanations. If we are making a single kind of prediction, then discarding the non-optimum models usually has a small penalty associated with it. However if we are working on a sequence of prediction problems, we will often find that the model that worked best in the past is inadequate for the new problems. When this occurs in science we have to revise our old theories. A good scientist will remember many theories that worked in the past but were discarded — either because they didn't agree with the new data, or because they were a priori "unlikely". New theories are characteristically devised by using failed models of the past, taking them apart, and using the parts to create new candidate theories. By having a large diverse set of (non-optimum) models on hand to create new trial models, ALP is in the best possible position to create new, effective models for prediction.

In the biological world, when a species loses its genetic diversity it can quickly succumb to small environmental changes — it soon becomes extinct. Lack of diversity in potato varieties in Ireland led to massive starvation.

When ALP is used in Genetic Programming, it's rich diversity of models can be expected to lead to very good, very fast solutions with little likelihood of "premature convergence".

Putting It All Together

I have described ALP and some of its properties, and to some extent, how it could be used in an AGI system. This section gives more details on how it works. We

start out with problems that are input/output pairs (I/O pairs). Given a sequence of them and a new input we want the machine to get a probability density distribution on its possible outputs. We allow I and O to be strings or numbers or mixtures of strings and numbers. Very many practical problems fall into this class — e.g. classification problems, identification problems, symbolic and numerical regression, grammar discovery... To solve the problems, we create trial functions that map inputs into possible outputs. We try to find several successful functions of this kind for each of the problems. It is usually desirable, though not at all necessary, to find a common function for all of the problems.

The trial functions are generated by a universal function language such as Lisp or Forth. We want a language that can be defined by a context free grammar, which we use to generate trial functions. The functions in such languages are represented by trees, which display the choices made in generating the trial functions. Each node in the tree represents a choice of a terminal or non-terminal symbol. Initially if there are k possible choices at a node, each choice is given probability $1/k$. These probabilities which we call “The Guiding Probability Distribution”(GPD) will evolve considerably, as the training sequence progresses.

In a simplified version of the technique we use Levin’s Search Procedure (Lsearch)¹ to find the *single* most likely solution to each of n problems. For each problem, I_i/O_i , we have a function, F_i represented in say, Reversed Polish Notation (RPN), such that $F_i(I_i) = O_i$. Using a suitable prediction or compression algorithm (such as Prediction by Partial Matching — PPM) we compress the set of function descriptions, $[F_i]$. This compressed version of the set of solutions can be predictive and enables us to get a probability distribution for the next sequence of symbols — giving a probability distribution over the next function, F_{n+1} . Levin Search gives us F_{n+1} candidates of highest probability first, so when we are given the next I_{n+1}/O_{n+1} pair, we can select the F_{n+1} of largest probability such that $F_{n+1}(I_{n+1}) = O_{n+1}$.

We then update the system by compressing the code for F_{n+1} into the previous sequence of solution functions and use this compressed code to find a solution to the next I/O in the training sequence. This continues until we have found solution functions for all of the problems in the sequence.

In a more realistic version of the system, using the diversity of ALP, we try to find several functions for each I/O pair in a corpus of say, 10 pairs. Suppose we have obtained 2 functions for each problem in the set. This amounts to $2^{10} = 1024$ different “codes” for the entire corpus. We then compress each of these codes and use the shortest say, 30 of them for prediction on the new input, I_{101} . The probability distribution on the output, O_{101} will be the weighted mean of the predictions of the

30 codes, weights being proportional to $2^{-\text{code length}}$. We also use these 30 codes to assign probabilities to grammar elements in constructing trial functions in future problems.

We mentioned earlier, the use of heuristic programming in designing this system. In both training sequence design and in the design and updating of the GPD, the techniques of heuristic programming are of much import.

Consider the problem of learning simple recursive functions. We are given a sequence of $n, F(n)$ pairs containing some consecutive values of n . We want to discover the function, $F()$. A heuristic programmer would try to discover how he himself would solve such a problem — then write a program to simulate himself.

For machine learning, we want to find a way for the machine to discover the trick used by the heuristic programmer — or, failing that, the machine should discover when to use the technique, or be able to break it down into subfunctions that are useful for other problems as well. Our continuing problem is to create training sequences and GPDs that enable the machine to do these things.

The Guiding Probability Distribution

The Guiding Probability Distribution (GPD) does two important things: first it discovers frequently used functions and assigns high probabilities to them. Second, it discovers for each function, contexts specifying the condition under which that function should be applied. Both of these operations are quantified by ALP.

In the previous section we described the GPD — how it made predictions of symbols in the next trial function — how the predictions were based on regularities in the sequence of symbols that represent in the solutions of earlier problems. Here we will examine the details of just how the GPD works.

Perhaps the simplest sequential prediction scheme is Laplace’s rule: The probability of the next symbol being A , say, is proportional to (the number of times A has occurred in the past, plus one). There is no dependency at all on context. This method of guiding probability evaluation was used in OOPS (Sch02) a system similar in several ways to the presently described system.

Prediction by Partial Matching (PPM) is a very fast, relatively simple, probability evaluation scheme that uses context very effectively. It looks at the string of symbols preceding the symbol to be predicted and makes a probability estimate on this basis.

PPM and variations of it have been among the best compressors in Hutter’s Entwiki challenge — a competition to discover the best compressor for 10^8 Bytes of the wikipedia.

Most of the improvements in PPM involve “context weighting” — they use several independent prediction schemes, each based on context in a different way. These systems are then merged by giving (localized) weights each of them.

¹See Appendix A

Merging of this sort can be used on the GPDs of several learning systems trained with different training sequences. A weakness of this simple merging is that the system does *not* create new functions by composing functions discovered by the different prediction schemes.

— A relevant quote from von Neumann — “For difficult problems, it is good to have 10 experts in the same room — but it is far better to have 10 experts in the same head”.

For really good induction, the GPD needs to recognize useful subfunctions and contexts to control the application of these subfunctions. PPM does this to some extent, but it needs much modification. While it is, in theory, easy to specify these modifications, it seems to be difficult to implement them with any speed. Much work needs to be done in this area.

Suppose Lsearch is generating candidate functions of I_j , the current input problem. If x is the part of the candidate that has been generated thus far, then in general, the probability distribution on the symbol to follow x , will be some function of I_j and x , i.e. $G(I_j, x)$. The form of G will slowly vary as we advance along the training sequence. For the best possible predictions, the form of G should be able to be any conceivable partial recursive function. Few prediction systems allow such generality.

How does the high compression obtained by PPM effect prediction systems? Compression ratio is the ratio of uncompressed string length to compressed string length. Compression ratio translates directly into increasing (geometric) mean probability of symbols and subsequences of symbols. A compression ratio of two increases probabilities to their square root. This translates directly into decreasing search times for solutions to problems. Using Lsearch, the time to find a particular solution will be about t_i/p_i , t_i being time needed to generate and test the solution, and p_i being the probability assigned to the solution. If $t_i = 10^{-6}$ seconds and $p_i = 10^{-16}$ for a particular uncompressed problem, then it will take about 10^{10} seconds — about 330 years to find a solution. A compression factor of two will increase p_i to the square root of 10^{-16} — i.e. 10^{-8} . So $t_i/p_i = 10^2$ seconds — about 1.7 minutes — a speed of up of 10^8 .

What compression ratios can we expect from PPM? A recent version got a compression ratio of 6.33 for a 71 kbyte LISP file. Unfortunately, much of this ratio was obtained by compressing long words used for function names. This kind of compression does not help find functional solutions. From the compression ratios of other less efficient compressors, my guess is that the elimination of this “long word” regularity would still give a compression ratio of greater than two. — Perhaps as much as three — enabling the rapid solution of problems that without compression would take many years of search time.

It is notable that high compression ratios were obtained for long text files. For us, the moral is that

we will not get full advantage of compression until our training sequences are long enough.

We have discussed PPM at some length as being a good initial GPD. A few other prediction methods that we have examined:

Genetic programming: Very effective. It can discover recursive prediction functions, but it is very slow. We have, however found many ways in which it could be sped up considerably (Sol06).

Echo State Machines (ESM). (JH04) A very deep neural net — very fast in implementation. Doesn’t do recursive functions, but we have found a way to move in that direction.

Support Vector Regression (SVR). (SS09) This is the application of SVMs to regression. The predictions are very good, but the algorithms are very slow and do not support recursion.

In any of the compressors, speed and compression ratios are both important. In selecting a compressor it is necessary to consider this trade-off.

Training Sequences

It is clear that the sequence of problems presented to the system will be an important factor in determining whether the mature system will be very much more capable than the infant system. The task of the training sequence is to teach functions of increasing difficulty by providing problems solved by these functions in a learnable progression. The main criterion of excellence in a training sequence: it enables the system to solve many problems outside the training sequence itself (out of sample data). To do this the problems in the sequence should be solved using a relatively small number of powerful functions. Designing training sequences of this sort is a crucial and challenging problem in the development of strong intelligence.

The system must also be able to recognize the context in which each function should be applied. This, however is a task for the guiding probability distribution.

In most ways, designing a training sequence for an intelligent machine is very similar to designing one for a human student. In the early part of the sequence, however, there is a marked difference between the two. In the early training sequence for a machine, we know *exactly* how the machine will react to any input problem. We can calculate a precise upper bound on how long it will take it to solve early problems. It is just

$$T_i/P_i \tag{1}$$

P_i is the probability that the machine assigns to the solution known by the trainer. T_i is the time needed to test that solution. I call this upper bound the “Conceptual Jump Size” (CJS). It tells us how hard a problem is for a particular machine — a measure of how long we expect that machine will take to solve it. I say “upper bound” because the system may discover a better, faster, solution than that known by the trainer.

This CJS estimate makes it easy to determine if a problem is feasible for a system at a particular point in its education. The P_i for a particular problem will vary during the life of the system, and for a properly constructed training sequence it should increase as the system matures. This increase in P_i can be used as a rough measure of the machines “rate of learning”.

Eventually in any training sequence for a very intelligent machine, the trainer will not be able to understand the system in enough detail to compute CJS values. The trainer then treats the machine as a human student. By noting which problems are easy and which are difficult for the machine, the trainer infers which relevant functions the machine has learned and which it has *not* learned and devises appropriate training problems.

Learning to train very intelligent machines should give useful insights on how to train human students as well.

There are at least two ways to write training sequences: “Bottom Up” and “Top Down”. The Bottom Up approach starts with some simple problems that have easy solutions in terms of the primitives of the reference language. The next problems in the sequence have solution functions that are simple combinations of functions the machine has already learned. This increase in complexity of problem solutions continues to the end of the training sequence.

In Top Down training sequence design, we start with a difficult problem that we know how to solve. We express its solution as a function mapping input to output. This function is then decomposed into simpler functions, and we design problems that are solvable by such functions. These functions are in turn factored into simpler functions and again we devise problems that are solvable by such functions. This breakup of functions and designing of problems continues until we reached the primitive functions of the reference language. The desired training sequence is the set of problems designed, but we present them in an order reversed from that in which they were invented.

The functions themselves form a partially ordered set. Function F_1 is *greater than* function F_2 if F_2 is used to create F_1 .

For more details on how to construct training sequences of this kind see (Sol89).

So far I’ve mainly worked on elementary algebra, starting with learning to evaluate algebraic expressions and solving simple equations — this can be continued with more complex equations, symbolic differentiation, symbolic integration etc. This list can go on to problems of arbitrary difficulty.

A promising source of training material: learning the definitions of the various operations in Maple and/or Mathematica.

Another possible source of training sequence ideas is “A Synopsis of Elementary Results in Pure and Applied Mathematics”, a book by G. S. Carr. It was the principal source of information about mathematics for

Ramanujan — one of the greatest creative geniuses of recent mathematics. His style was one of imaginative inductive leaps and numerical checking — much in the manner of how we would like the present system to operate.

After the machine has an understanding of algebra, we can train it to understand English sentences about algebra. This would not include “word problems” which typically require knowledge about the outside world.

It cannot be emphasized too strongly, that the goal of early training sequence design, is *not* to solve hard problems, but to get problem solving information into the machine. Since Lsearch is easily adapted to parallel search, there is a tendency to try to solve fairly difficult problems on inadequately trained machines. The success of such efforts is more a tribute to progress in hardware design than to our understanding and exploiting machine learning.

In Conclusion

We have a method for designing training sequences. We have a method for updating the guiding probability distribution. We have a method for detecting/measuring “learning” in the system.

These three techniques are adequate for designing a true AGI.

If the system does not learn adequately, the fault is in either the training sequence (the conceptual jumps needed are too large) — or that the update algorithm may not be able to recognize important kinds of functions that occur in the training sequence and know under what conditions they should be used — in which case we must modify the training sequence and/or the update algorithm. The update algorithm must be designed so that it can readily spot functions used in the past that are relevant to current problems.

What we have is a recipe for training an intelligence system, and a few ways to debug our attempts at training it. The system itself is built on ALP, which is certainly adequate to the task. Our understanding of much of our own human learning will probably be inadequate at first. There will be conceptual leaps in the training sequences which we don’t know how to break down into smaller jumps. In such cases it may be necessary to practice a bit of “Brain Surgery” to teach the machine — direct programming of functions into the reference language. Usually we will try to avoid such drastic measures by simplification of problems or by giving auxiliary related problems — by giving “hints”.

We have mentioned the possibility of merging the guiding probability distributions of different systems created by independent investigators. It would be well if there were several research groups working on systems of the type described, with enough similarity in the reference languages and update algorithms so that the guiding probability distributions could, indeed, be merged.

The system we have described will do fairly general kinds of prediction. It can be regarded as Phase 1 of

a larger project. Phase 2 (Sol03) — is built on Phase 1 and is designed to solve even more general kinds of problems. In particular, it is able to work time-limited optimization problems — for example, “Get as good a solution as you can to this traveling salesman problem in 10 minutes”. Most practical problems in science and engineering are of this kind. This includes the problem of improving the GPD of Phase 1 — enabling the system to significantly improve itself.

Appendix A: Levin’s Search Procedure

It would seem that evaluating a very large number of functions would take an enormous amount of time. However, by using a search technique similar to one used by Levin for a somewhat different kind of problem, it is possible to perform the search for acceptable functions in something approaching optimum speed. It may occasionally take a long time to find a very good solution — but it is likely that no other search technique with equivalent education and hardware could have found that solution any faster.

How the procedure works: Suppose we have an input string, I_1 and an output string, O_1 . We also have a probability distribution p_j over all possible functions, F_j and we want to find high probability functions, F_j , such that $F_j(I_1) = O_1$.

We could simply apply many random functions to I_1 and watch for functions that meet our requirements. This would take a lot of time. There is, however, a much more efficient way:

We select a small time limit, T , and we test all functions, F_j such that

$$t_j < Tp_j \quad (2)$$

Here p_j is the probability of the function being tested, and t_j is the time required to test it. The test itself is to see if $F_j(I_1) = O_1$. If we find no function of this sort, we double T and go through the same test procedure. We repeat this routine until we find satisfactory functions. If F_j is one of the successful functions, then the entire search for it will take time $\leq 2t_j/p_j$.

There is a faster, time shared version of Lsearch that takes only time $\leq t_j/p_j$, but it takes much, much more memory.

An important feature of Lsearch is that it is able to deal with trials that do not converge — i.e. for $t_j = \infty$

Appendix B: Frustrating Computable Probability Distributions

Given a computable probability function μ , I will show how to generate a deterministic sequence (i.e. probabilities are only 0 and 1)

$$Z = Z_1 Z_2 Z_3 \dots$$

to which μ gives probabilities that are *extremely bad*: they are always in error by $\geq .5$.

Let $\mu(Z_{n+1} = 1 | Z_1 \dots Z_n)$ be μ ’s estimate that Z_{n+1} will be 1, in view of $Z_1 \dots Z_n$.

$$\text{if } \mu(Z_1 = 1 | \bigwedge) < .5 \text{ then } Z_1 = 1 \text{ else } Z_1 = 0$$

$$\begin{aligned} &\text{if } \mu(Z_2 = 1 | Z_1) < .5 \text{ then } Z_2 = 1 \text{ else } Z_2 = 0 \\ &\text{if } \mu(Z_k = 1 | Z_1, Z_2, \dots, Z_{k-1}) < .5 \text{ then } Z_k = 1 \text{ else } \\ &Z_k = 0. \end{aligned}$$

In a similar way, we can construct probabilistic sequences in which μ is in error by $\geq \epsilon$, where, ϵ can have any value between 0 and .5.

References

- P. Gács. Theorem 5.2.1. In *An Introduction to Kolmogorov Complexity and Its Applications*, pages 328–331. Springer-Verlag, N.Y., second edition, 1997. by Li, M. and Vitányi, P.
- M. Hutter. Optimality of universal bayesian sequence prediction for general loss and alphabet. Technical report, IDSIA, Lugano, Switzerland, 2002. <http://www.idsia.ch/~marcus/ai/>.
- H. Jaeger and H. Haas. Harnessing nonlinearity: Predicting chaotic systems and saving energy in wireless communication. *Science*, Vol. 304(5667):78–80, April 2004.
- J. Poland and M Hutter. MDL convergence speed for Bernoulli sequences. *Statistics and Computing*, 16:161–175, 2006.
- J. Rissanen. Modeling by the shortest data description. *Automatica*, 14:465–471, 1978.
- J. Schmidhuber. Optimal ordered problem solver. TR Idsia-12-02, IDSIA, Lugano, Switzerland, July 2002. <http://www.idsia.ch/~juergen/oops.html>.
- R.J. Solomonoff. Complexity-based induction systems: Comparisons and convergence theorems. *IEEE Trans. on Information Theory*, IT-24(4):422–432, 1978.
- R.J. Solomonoff. A system for incremental learning based on algorithmic probability. In *Proceedings of the Sixth Israeli Conference on Artificial Intelligence, Computer Vision and Pattern Recognition*, pages 515–527, Tel Aviv, Israel, December 1989.
- R.J. Solomonoff. Progress in incremental machine learning. TR Idsia-16-03, IDSIA, 2003. Given at NIPS Conference, Dec. 14, 2002, Whistler, B.C., Canada.
- R.J. Solomonoff. Machine learning - past and future. Dartmouth, N.H., July 2006. Lecture given in 2006 at AI@50, The Dartmouth A. I. Conference: The Next Fifty Years.
- N. Sapankevych and R. Sankar. Time series prediction using support vector machines: A survey. *IEEE Computational Intelligence*, Vol. 4(2):24–38, May 2009.
- C.S Wallace and D.M. Boulton. An information measure for classification. *The Computer Journal*, 11:185–194, 1968.

All of Solomonoff’s papers and reports listed here are available at <http://world.std.com/~rjs/pubs.html>

Frontier Search

Sun Yi, Tobias Glasmachers, Tom Schaul, and Jürgen Schmidhuber

IDSIA, University of Lugano
Galleria 2, Manno, CH6928, Switzerland

Abstract

How to search the space of programs for a code that solves a given problem? Standard asymptotically optimal Universal Search orders programs by Levin complexity, implementing an exponential trade-off between program length and runtime. Depending on the problem, however, sometimes we may have a good reason to greatly favor short programs over fast ones, or vice versa. *Frontier Search* is a novel framework applicable to a wide class of such trade-offs between program size and runtime, and in many ways more general than previous work. We analyze it in depth and derive exact conditions for its applicability.

Introduction

In an inversion problem, the aim is to find a program p that produces a desired output x . Algorithms that search the space of programs for p are guided (implicitly or explicitly) by an optimality criterion, which is generally based on program length and runtime. Levin complexity, a criterion where the trade-off between program length and runtime is exponential, can readily be optimized using Levin Search (Lev73). The framework of ‘speed priors’ (Sch02) results in a more flexible search scheme. The aim of this paper is to develop a search scheme applicable to an even wider class of user-defined optimality criteria.

More formally, consider a programming language L and a (countable) set \mathcal{P} of programs. Let $p : \mathbb{N} \rightarrow \mathcal{P}$ be an enumeration of \mathcal{P} . We refer to the i -th program as p_i . Then, Levin Search finds $p \in \mathcal{P}$ such that $L(p) = x$. It works by executing in parallel all programs in \mathcal{P} such that the fraction of time allocated to the i -th program is $2^{-l(p_i)}/S$, where $l(p_i)$ is the length of a *prefix-free* binary encoding of p_i , and $0 < S \leq 1$ is a normalization constant. Alternatively, a growing number of programs can be executed for a fixed exponentially growing time one after the other, which involves restarting the programs several times. This simpler algorithm performs worse only by a constant factor.

Levin Search, though simple in its form, enjoys two important theoretical properties. The first property concerns *the time required to find a solution*. It is guaranteed that Levin Search solves the inversion problem within time $2^{l(p^*)+1} \cdot S \cdot \tau(p^*)$, where $p^* \in \mathcal{P}$ is the fastest program that solves the problem, and $\tau(p^*)$ is the number of time steps after which p^* halts. Since p^* depends solely on the problem

itself, one can claim that Levin Search solves the problem in time linear to the runtime of the fastest program available, despite the prohibitively large multiplicative constant.

The second property, on the other hand, characterizes *the quality of the solution*. It has been shown that the program found by Levin Search (asymptotically) optimizes the Levin complexity K_l defined as

$$K_l(x) = \min_{p \in \mathcal{P}} \{l(p) + \log \tau(p) \mid L(p) = x\},$$

which is a computable, time-bounded version of the Kolmogorov complexity (LV93). Note that in this paper, all logarithms are to base 2.

Whereas the linear time bound property of Levin Search receives considerable appreciation, less attention is paid to the quality of the solution. In general, solution quality is measured by the complexity function. Thus, a particular search scheme such as Levin Search implies a complexity function it (asymptotically) minimizes. In this paper we approach the problem from the other end, assuming that a complexity function is given, but not a search scheme. The central question asked in this paper is:

*Given a certain optimality criterion,
how do we search the space of programs?*

The remainder of the paper is structured as follows. First we discuss the space of possible complexity criteria, then we introduce our algorithm, Frontier Search, and give exact conditions on its applicability. We find that this approach allows for optimality criteria that are more flexible than the speed prior. Finally we present an approximation to Frontier Search that achieves asymptotically constant overhead complexity.

Generalized Complexity Criteria

Let us first focus on the form of K_l . Assume both p_1 and p_2 solve the problem $L(p) = x$ and achieve the same value of $l(p) + \log(\tau(p))$. If p_1 is m bits shorter than p_2 , the execution time of p_1 would be 2^m times larger than for p_2 . This encodes an inherent trade-off between the program execution time and its length, namely, *how much more time we are willing to invest for finding a solution which is 1 bit shorter*. In the remainder of this paper we replace the concept of program length with program order in the sense

of the enumeration $p : \mathbb{N} \rightarrow \mathcal{P}$. The familiar length encoding can be recovered by enumerating programs by increasing length.

Now consider the following three scenarios:

1. We are trying to find a relatively tight upper bound on the Kolmogorov complexity of a string x . This amounts to finding a concise representation for a given x , and the length of the program found matters much more than its execution time. In this case, we might choose a different complexity criterion instead of K_i which emphasizes the program length more, for example,

$$K_1(x) = \min \{ [l(p)]^s + \log(\tau(p)) \mid L(p) = x \}$$

with $s > 1$. (In the limit $s \rightarrow \infty$ we get Kolmogorov complexity (LV93). Unfortunately, it is incomputable.)

2. We are searching for a representation of x which is assumed to be used a lot in the future, amounting to executing the resulting program p regularly. We may argue that quicker programs are preferred despite their slightly longer length since they will be executed often. In this case, the complexity criterion

$$K_2(x) = \min \{ [l(p)]^{1/s} + \log(\tau(p)) \mid L(p) = x \}$$

with $s > 1$, which favours quicker programs, makes more sense.

3. We have prior knowledge telling us that programs with a certain structure (in the simplest case, programs of a certain length) should be preferred, and we would like to encode such knowledge into the complexity criterion. An extreme example is that we do not want to run programs of trivial length (e.g., $l(p) = 1$) for half of the total running time as suggested in Levin Search. (Certainly, such prior knowledge can be incorporated into the programming language itself, but that necessitates re-designing the language every time we vary the requirement (SS10).)

All these scenarios call for a more general approach: We want our search to respect a complexity criterion suitable for the problem at hand. *Starting from a complexity criterion which encodes the desired trade-off between execution time and program order, we build up a search algorithm that finds the optimal solution in the sense of the given complexity criterion.* The search algorithm should be invariant w.r.t. any monotonically increasing (i.e., order preserving) transformation of the complexity criterion, since the program minimizing $l(p) + \log(\tau(p))$ would also minimize $2^{l(p)} \cdot \tau(p)$, or in general, $f(l(p) + \log(\tau(p)))$ for any monotonically increasing function $f : \mathbb{R} \rightarrow \mathbb{R}$.

Our answer to the problem above is a simple search algorithm called *Frontier Search*. It maintains a ‘frontier’ of the possible execution steps and at each iteration selects the one minimizing the given complexity criterion. We prove that under reasonable technical constraints on the complexity criterion Frontier Search indeed finds the optimal program. Also, we show the connection between Frontier Search and Levin Search, as well as universal search with ‘speed prior’ (Sch02), and demonstrate that Frontier Search is more general since it allows the encoding of speed preferences which cannot be represented using the speed prior approach.

Frontier Search

We consider the general complexity criterion

$$K_\psi(x) = \min_{i \in \mathbb{N}} \{ \psi(i, \tau_i) \mid L(p_i) = x \},$$

where τ_i is the execution time of p_i , and $\psi : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{R}$ is a complexity function encoding the trade-off between program length and execution time. For example, for the choice $\psi(i, \tau) = 2^i \cdot \tau$, we recover Levin Search under the trivial encoding $p_i = 1 \dots 10$ (i ones, one zero). Furthermore, $\psi(i, \tau) = \tau/\pi_i$, with $\pi_i > 0$ and $\sum_{i \in \mathbb{N}} \pi_i = 1$, encodes universal search based on the speed prior π (Sch02).

Algorithm 1 presents the pseudocode for Frontier Search, and Figure 1 illustrates its operation. The set $\{(i, \tau_i) \mid i \in \{1, \dots, n\}\} \subset \mathbb{N} \times \mathbb{N}$ with $\tau_n = 1$ forms the current ‘frontier’, i.e., available executions in the next time step. If program p_n gets executed, then the frontier automatically expands to include a new program p_{n+1} . We assume that for multiple j minimizing $\psi(j, \tau_j + 1)$ the smallest j is chosen.

Algorithm 1: Frontier Search.

Input: ψ, L, x, \mathcal{P}

Output: $p \in \mathcal{P}$ such that $L(p) = x$

$n \leftarrow 1$;

$\tau_n \leftarrow 0$;

while true do

$i \leftarrow \arg \min \{ \psi(j, \tau_j + 1) \mid j \in \{1, \dots, n\} \}$;

execute p_i for 1 step;

if p_i halts and $L(p_i) = x$ **then return** p_i ;

$\tau_i \leftarrow \tau_i + 1$;

if $i = n$ **then**

$n \leftarrow n + 1$;

$\tau_n \leftarrow 0$;

end

end

The following definitions will prove handy for the analysis of Frontier Search.

Definition 1. *Formally, the set of all possible frontiers is given by*

$$\mathcal{F} = \{ \{(1, \tau_1), \dots, (n-1, \tau_{n-1}), (n, 1)\} \mid n \in \mathbb{N} \text{ and } \tau_i \in \mathbb{N} \ \forall i \in \{1, \dots, n-1\} \} \cong \bigcup_{n \in \mathbb{N}} \mathbb{N}^{n-1}.$$

For a given frontier $F = \{(1, \tau_1), \dots, (n, 1)\} \in \mathcal{F}$ we say that the grid points $(i, \tau) \in F$ are *on* the frontier, points (i, τ) with $i < n$ and $\tau < \tau_i$ are *inside* the frontier, and all other grid points are *outside* the frontier, see also Figure 1. The points inside the frontier correspond to the program steps already executed by Frontier Search in order to reach the current frontier.

For any given frontier there exists a complexity function ψ that makes Frontier Search indeed reach this frontier. A simple choice is to set ψ to $1/2$ for all points inside the frontier, and to $i + \tau$ for all other points.

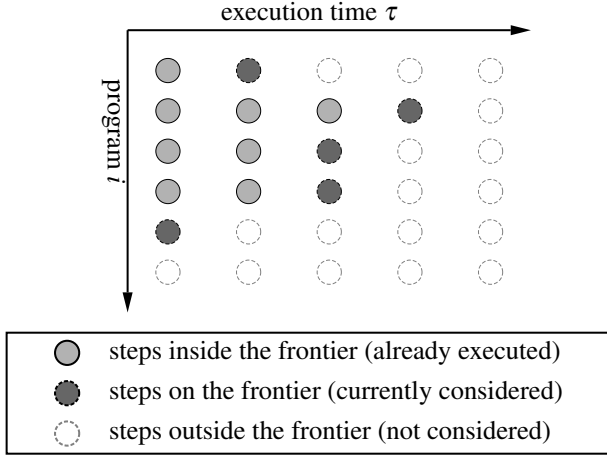


Figure 1: Illustration of Frontier Search. In each iteration, a frontier of possible execution steps $(i, \tau_i + 1)$, i.e., executing the $(\tau_i + 1)$ -th command of program p_i , is maintained. The step which minimizes $\psi(i, \tau_i + 1)$ is executed.

Definition 2. We define the partial order relation

$$\{(1, \tau_1), \dots, (n, 1)\} \leq \{(1, \tau'_1), \dots, (n', 1)\} \\ \Leftrightarrow n \leq n' \text{ and } \tau_i \leq \tau'_i \text{ for all } i \in \{1, \dots, n\}$$

on the set \mathcal{F} of frontiers.

In this canonical order relation it holds $F \leq F'$ if and only if the points inside F are a subset of the points inside F' . Thus, for each complexity function ψ Frontier Search generates a strictly growing sequence $(F_t^\psi)_{t \in \mathbb{N}}$ of frontiers.

Definition 3. For a frontier $F = \{(1, \tau_1), \dots, (n, 1)\} \in \mathcal{F}$ we define the time $T(F) = \sum_{i=1}^{n-1} (\tau_i - 1)$ necessary for frontier search to reach this frontier.

The identity $T(F_t^\psi) = t$ is obvious.

Definition 4. Assume step $\tau + 1$ of program p_i is executed by Frontier Search with complexity function ψ in finite time. Then we associate the frontier

$$F_{(i, \tau)}^\psi = \max \{F_t^\psi \mid t \in \mathbb{N} \text{ and } (i, \tau) \in F_t^\psi\}$$

with the tuple (i, τ) .

Let us introduce a useful auxiliary property of complexity functions:

Definition 5. We say that a complexity function $\psi : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{R}$ is frontier-bounded if for any $(i, \tau) \in \mathbb{N} \times \mathbb{N}$ there exist $n > i$ and $(\tau_1, \dots, \tau_{i-1}, \tau_{i+1}, \dots, \tau_{n-1}) \in \mathbb{N}^{n-2}$, such that

$$\begin{aligned} \psi(j, \tau_j) &> \psi(i, \tau) \quad \forall j \in \{1, \dots, i-1\} \\ \psi(j, \tau_j) &\geq \psi(i, \tau) \quad \forall j \in \{i+1, \dots, n-1\} \\ \psi(n, 1) &\geq \psi(i, \tau) \end{aligned}$$

Note that only the first of the three inequalities is strict. Intuitively, the definition states that for each (i, τ) there exists a frontier $\{(i, \tau_i) \mid i \in \{1, \dots, n\}\} \ni (i, \tau)$ containing this tuple, leading to the execution of step τ of program p_i in the next iteration.

The following statement provides us with two simple criteria implying frontier-boundedness.

Proposition 6. A complexity function $\psi : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{R}$ fulfilling one of the properties

$$\begin{aligned} |\{(i', \tau') \in \mathbb{N} \times \mathbb{N} \mid \psi(i', \tau') \leq \psi(i, \tau)\}| &< \infty \\ \forall (i, \tau) \in \mathbb{N} \times \mathbb{N} \end{aligned} \quad (1)$$

or

$$\begin{aligned} \lim_{i \rightarrow \infty} \psi(i, 1) &= \infty \\ \text{and } \lim_{\tau \rightarrow \infty} \psi(i, \tau) &= \infty \quad \forall i \in \mathbb{N} \end{aligned} \quad (2)$$

is frontier-bounded.

Proof. Case (1): For fixed $(i, \tau) \in \mathbb{N} \times \mathbb{N}$ consider the set $S = \{(i', \tau') \in \mathbb{N} \times \mathbb{N} \mid \psi(i', \tau') \leq \psi(i, \tau)\}$. We define $n = 1 + \max\{i \in \mathbb{N} \mid \exists \tau \in \mathbb{N} \text{ such that } (i, \tau) \in S\}$ as well as $\tau_i = 1 + \max\{\tau \in \mathbb{N} \text{ such that } (i, \tau) \in S\}$ for all $i \in \{1, \dots, i-1\} \cup \{i+1, \dots, n-1\}$. All maxima exist because S is finite per assumption, and using the convention $\max(\emptyset) = 0$.

Case (2): Again we fix $(i, \tau) \in \mathbb{N} \times \mathbb{N}$. From $\lim_{n \rightarrow \infty} \psi(n, 1) = \infty$ we conclude that there exists $n \in \mathbb{N}$ such that $\psi(n, 1) \geq \psi(i, \tau)$. Now for all $j \in \{1, \dots, i-1\} \cup \{i+1, \dots, n-1\}$ we have $\lim_{\tau_j \rightarrow \infty} \psi(j, \tau_j) = \infty$, from which we conclude the existence of τ_j such that $\psi(j, \tau_j) \geq \psi(i, \tau)$.

By construction in both cases the frontier size $n \in \mathbb{N}$ and the tuples $(\tau_1, \dots, \tau_{i-1}, \tau_{i+1}, \dots, \tau_{n-1})$ fulfill the conditions of Definition 5. \square

The next proposition clarifies the significance of frontier-boundedness, namely that this property guarantees that Frontier Search executes every program for sufficiently many steps.

Proposition 7. Frontier Search applied to a complexity function ψ executes program p_i for τ steps in finite time for all $(i, \tau) \in \mathbb{N} \times \mathbb{N}$ iff ψ is frontier-bounded.

Proof. (\Leftarrow) Assume ψ is frontier-bounded and fix $(i, \tau) \in \mathbb{N} \times \mathbb{N}$. Then we define $n = \min\{n' \in \mathbb{N} \mid n' > i \text{ and } \psi(n', 1) \geq \psi(i, \tau)\}$, which is well-defined due to the frontier-boundedness of ψ . Accordingly we define $\tau_j = \min\{\tau' \in \mathbb{N} \mid \psi(j, \tau') > \psi(i, \tau)\}$ for each $j \in \{1, \dots, i-1\}$ and $\tau_j = \min\{\tau' \in \mathbb{N} \mid \psi(j, \tau') \geq \psi(i, \tau)\}$ for all $j \in \{i+1, \dots, n-1\}$, which are all well-defined (none of the arguments of the min-operator is empty) due to ψ being frontier-bounded. When starting from the corresponding frontier $F = \{(1, \tau_1), \dots, (n-1, \tau_{n-1}), (n, 1)\}$, Frontier Search executes program p_i in the next step. Obviously it is impossible for Frontier Search to pass any point of this frontier without executing (i, τ_i) . The search can spend only $T(F) < \infty$ steps before reaching this frontier. Thus, step τ_i of program p_i is executed in step $T(F) + 1 < \infty$. As an aside, this argument shows $F = F_{(i, \tau)}^\psi$. In other words, the frontier $F_{(i, \tau)}^\psi$ associated with (i, τ) can be constructed as described above.

(\Rightarrow) For $(i, \tau) \in \mathbb{N} \times \mathbb{N}$ let $F_{(i, \tau)}^\psi = \{(1, \tau_1), \dots, (n, 1)\}$ denote the associated frontier. Then n and $(\tau_1, \dots, \tau_{i-1}, \tau_{i+1}, \dots, \tau_{n-1})$ per construction fulfill the requirements of Definition 5. \square

Corollary 8. Assume there exists $p \in \mathcal{P}$ with $L(p) = x$. Then Frontier Search applied to a frontier-bounded complexity function ψ halts.

If ψ is not frontier-bounded some steps never get executed. Let us have a look at two illustrative counter examples: First consider $\psi(i, \tau) = \tau - 1/i$. This complexity function is not frontier-bounded since for all $n \in \mathbb{N}$ we have $\psi(n, 1) = 1 - 1/n < 1 = \psi(1, 2)$. In this case, Frontier Search executes every program only for a single step. Second, consider $\psi(i, \tau) = i - 1/\tau$, which is not frontier-bounded since $\psi(1, \tau) < \psi(2, 1)$ for all $\tau \in \mathbb{N}$. With this ψ , Frontier Search executes the first program forever (provided that it doesn't halt). The same behavior results for constant $\psi(i, \tau)$, or for $\psi(i, \tau) = l(p_i)$ corresponding to Kolmogorov complexity.

Assume $\psi(i, \tau)$ is non-decreasing in τ . Intuitively, the proceeding of frontier search can be understood by a mechanical picture. Consider a landscape on top of the positive quadrant of the plane, with grid altitude profile given by ψ . The execution of Frontier Search amounts to flooding water into the landscape at the origin, such that exactly one integer square is flooded in each iteration, corresponding to the next program step executed. See Figure 2 for an illustration.

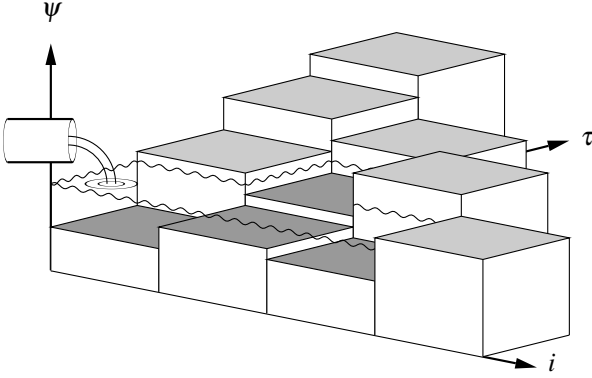


Figure 2: The operation of Frontier Search can be thought of as filling water into the landscape given by $\psi(i, \tau)$, which in this case must be monotonic in τ .

Since ψ serves as a complexity criterion, it is reasonable to assume that quicker programs are always preferred. If possible, we will further assume that programs are pre-ordered by complexity. This leads us to the definition of two handy conditions on complexity functions ψ :

Definition 9. We say that a complexity function ψ is proper if it is frontier-bounded and fulfills the monotonicity conditions

$$\begin{aligned} \psi(i, \tau) &\leq \psi(i, \tau + 1) \quad \forall (i, \tau) \in \mathbb{N} \times \mathbb{N} \\ \text{and } \psi(i, 1) &\leq \psi(i + 1, 1) \quad \forall i \in \mathbb{N}. \end{aligned}$$

We call a complexity function separable if it is frontier-bounded and is of the form $\psi(i, \tau) = \eta_i \cdot \tau$ with $\eta_i > 0$ and $\eta_i \leq \eta_{i+1}$ for all $i \in \mathbb{N}$.

A few notes are in order. First, it is easy to see that separability implies properness. Second, a separable complexity function is equivalent, for example, to one of the form

$\psi(i, \tau) = \log(\eta_i) + \log(\tau)$ (or any other monotonic transformation). However, in the following we will stick to the multiplicative form, which has a straight forward interpretation: We fix the same cost η_i for all execution steps of program p_i . This turns $\pi_i = 1/\eta_i$ into a (non-normalized) prior over the space \mathcal{P} of programs. For example, Levin Search induces the prior $\pi_i = 2^{-l(p_i)}$. In contrast to speed prior-based search, the prior $\pi_i = 1/\eta_i$ may be improper¹ without distorting Frontier Search in any way. This makes Frontier Search widely applicable, even in the restricted case of separable complexity functions.

While we already know that frontier-boundedness makes sure that Frontier Search finds a solution to the problem (if one exists), this property is not sufficient to guarantee optimality. Here, properness comes into play:

Proposition 10. We consider Frontier Search with proper complexity function ψ . Then Frontier Search finds the solution which minimizes ψ . Let i^* be the index of the minimizing program solving the problem in τ^* steps, then the total number of steps $T(F_{(i^*, \tau^*)}^\psi) + 1$ executed by Frontier Search is given by

$$\begin{aligned} &|\{(i, \tau) \in \mathbb{N} \times \mathbb{N} \mid \psi(i, \tau) < \psi(i^*, \tau^*)\}| \\ &+ |\{(i, \tau) \in \mathbb{N} \times \mathbb{N} \mid \psi(i, \tau) = \psi(i^*, \tau^*) \text{ and } i \leq i^*\}|. \end{aligned}$$

Proof. Consider the last frontier $F_{(i^*, \tau^*)}^\psi = \{(1, \tau_1), \dots, (n-1, \tau_{n-1}), (n, 1)\}$ before executing the final statement of p_{i^*} . In this moment we have $\psi(i, \tau_i) \geq \psi(i^*, \tau^*)$ for all points on the frontier. Now the monotonicity ensures $\psi(i', \tau') \geq \psi(i', \tau_i) \geq \psi(i, \tau_i) \geq \psi(i^*, \tau^*)$ for all $i' \in \{1, \dots, n\}$ and $\tau' > \tau_i$, and $\psi(i', \tau') \geq \psi(i', 1) \geq \psi(n, 1) \geq \psi(i, \tau)$ for all $i' > n$ and $\tau \in \mathbb{N}$. Thus, all points outside the frontier have complexity larger or equal to $\psi(i, \tau)$, independently of whether they solve the problem or not. On the other hand, all points inside the frontier have complexity values of at most $\psi(i^*, \tau^*)$. But all steps corresponding to these points have been executed without solving the problem and halting.

The number of steps follows from the first statement, just notice that we assume that the program with smaller index is selected whenever two steps achieve the same complexity. \square

Corollary 11. We consider Frontier Search with proper complexity function ψ . Let i^* be the index of the minimizing program solving the problem in τ^* steps. Then the total number of steps is bounded by

$$\begin{aligned} &|\{(i, \tau) \in \mathbb{N} \times \mathbb{N} \mid \psi(i, \tau) < \psi(i^*, \tau^*)\}| \\ &\leq T(F_{(i^*, \tau^*)}^\psi) < T(F_{(i^*, \tau^*)}^\psi) + 1 \leq \\ &|\{(i, \tau) \in \mathbb{N} \times \mathbb{N} \mid \psi(i, \tau) \leq \psi(i^*, \tau^*)\}| \end{aligned}$$

Now we can effectively bound the total number of steps executed by Frontier Search for any given proper complexity criterion. We demonstrate three important cases:

¹The prior π is proper if it can be normalized to sum to one, i.e., if $\sum_{i \in \mathbb{N}} \pi_i < \infty$.

Example 12. Consider the criterion in the Levin complexity $\psi(i, \tau) = 2^i \cdot \tau$. This function is separable with prior $\pi_i = 2^{-i}$. If program i halts after running τ steps, the total execution time is upper bounded by

$$\begin{aligned} T(F_{(i, \tau)}^\Psi) &\leq |\{(i', \tau') \in \mathbb{N} \times \mathbb{N} \mid 2^{i'} \cdot \tau' \leq 2^i \cdot \tau\}| \\ &= |\{(1, \tau') \mid \tau' \leq 2^{i-1} \tau\}| \\ &\quad + |\{(2, \tau') \mid \tau' \leq 2^{i-2} \tau\}| \\ &\quad + \dots + |\{(i, \tau') \mid \tau' \leq \tau\}| \\ &\quad + |\{(i+1, \tau') \mid \tau' \leq \frac{\tau}{2}\}| + \dots + 1 \\ &\leq 2^{i-1} \tau + 2^{i-2} \tau + \dots + \tau + \left\lfloor \frac{\tau}{2} \right\rfloor + \left\lfloor \frac{\tau}{4} \right\rfloor + \dots + 1 \\ &\leq 2^i \tau \in O(\tau) . \end{aligned}$$

So the total execution time is linear in τ . The same calculation works for arbitrary proper speed priors π_i .

Example 13. As a second example we consider the separable criterion $\psi(i, \tau) = i \cdot \tau$, which corresponds to the improper prior $\pi_i = 1/i$. Again, let program p_i halt after τ steps. The number of steps for Frontier Search to execute is bounded by

$$\begin{aligned} T(F_{(i, \tau)}^\Psi) &\leq |\{(i', \tau') \in \mathbb{N} \times \mathbb{N} \mid i' \cdot \tau' \leq i \cdot \tau\}| \\ &\leq i \tau \cdot \log(i \tau) \in O(\tau \cdot \log(\tau)) , \end{aligned}$$

which may still be considered affordable.

Example 14. Last but not least we consider the complexity function $\psi(i, \tau) = \tau \cdot (i + \tau)$, which puts a strong emphasis on short execution time. It is proper, but not separable, because it increases the penalty per step the longer a program runs. Let program p_i halt after τ steps, and let $c = \psi(i, \tau) = \tau \cdot (i + \tau)$ be its complexity. Then the total number of steps executed is lower bounded by

$$\begin{aligned} T(F_{(i, \tau)}^\Psi) &\geq |\{(i', \tau') \in \mathbb{N} \times \mathbb{N} \mid \tau' \cdot (i' + \tau') < c\}| \\ &= \left| \left\{ (1, \tau') \mid \tau' < \frac{-1 + \sqrt{1 + 4c}}{2} \right\} \right| \\ &\quad + \left| \left\{ (2, \tau') \mid \tau' < \frac{-2 + \sqrt{4 + 4c}}{2} \right\} \right| \\ &\quad + \dots + \left| \left\{ (l, \tau') \mid \tau' < \frac{-l + \sqrt{l^2 + 4c}}{2} \right\} \right| , \end{aligned}$$

where

$$l = \arg \min_k \left\{ \frac{-k + \sqrt{k^2 + 4c}}{2} \leq 2 \right\} \Rightarrow l \geq \frac{c}{2} - 2 .$$

So when c is sufficiently large,

$$\begin{aligned} &|\{(i', \tau') \in \mathbb{N} \times \mathbb{N} \mid \tau' \cdot (i' + \tau') < c\}| \\ &\geq \sum_{k=1}^{\frac{c}{2}-2} \frac{-k + \sqrt{k^2 + 4c}}{2} \\ &\geq \left(\frac{c}{2} - 2 \right) \frac{2 - \frac{c}{2} + \sqrt{\frac{c^2}{4} + 4 + 3c}}{2} \in \Omega(c) = \Omega(\tau^2) . \end{aligned}$$

Thus, the search requires $\Omega(\tau^2)$ steps.²

Reduction of Overhead Complexity

Algorithm 1 has one serious drawback compared to plain Levin Search: The ‘arg min’-operation used to decide which program to execute next takes at least $\log(n)$ operations (using efficient data structures), where n is the size of the current frontier. This growing overhead, compared to the constant time spent on executing the underlying programs, is unsatisfactory, because asymptotically the fraction of time spent on program execution tends to zero. In this section we provide an algorithm that, under reduced requirements, achieves an amortized constant overhead.

Instead of strictly minimizing the complexity function ψ in each iteration we weaken the requirements as follows:

- We consider separable complexity functions. Furthermore, we assume that η_i is available in a binary encoding.
- The minimization of the complexity function may be only approximate. Let τ_i denote the position of the current frontier for program p_i , and let $\tilde{\tau}_i$ be the number of steps actually executed. Then we require $\lim_{\tau_i \rightarrow \infty} \tilde{\tau}_i / \tau_i = 1$ for all $i \in \mathbb{N}$.
- The complexity function does not need to be minimized in each single iteration. Instead, we ask for a growing sequence $(t_n)_{n \rightarrow \infty}$ of iterations in which the current frontier approximately minimizes the complexity function.

Approximate Frontier Search is introduced in Algorithm 2. It approximates Frontier Search in the above sense. The algorithm runs in epochs, maintaining a growing target complexity C . In each epoch it executes all programs with single-step complexity $\eta_i \leq C / \lceil \log(C) \rceil^2$ until they reach the target complexity, or in other words the frontier $\psi \approx C$. The frontier is approximated by delaying the execution of programs with relatively high single-step complexity $\eta_i > C / \lceil \log(C) \rceil^2$. It is easy to see that Approximate Frontier Search indeed fulfills the conditions listed above. As soon as $C / \lceil \log(C) \rceil^2$ (which tends to infinity) exceeds η_i the condition $\tilde{\tau}_i = \tau_i$ is fulfilled for the sequence $(t_e)_{e \in \mathbb{N}}$ of iterations finishing epochs.

In the following we analyze the complexity of the overhead created by Algorithm 2.

Proposition 15. The number of operations of Algorithm 2 in between executing two program steps is upper bounded by a constant in an amortized analysis.

Proof. We need a few basic facts about operations on binary encoded numbers. Recall that adding a constant value to a variable takes amortized constant time. Therefore counting in a loop from 1 to m takes $O(m)$ time. Computing $a + b$ takes $O(\min\{\log(a), \log(b)\})$ operations, and so does the comparison $a < b$. The multiplication $a \cdot b$ requires $O(\log(a) \cdot \log(b))$ operations, and an integer division $\lfloor a/b \rfloor$ costs $O(\log(a/b) \cdot \log(b)) \leq O((\log(a))^2)$ computational time. The computation of $\lceil \log(a) \rceil$ can be performed in at most $O(\log(a))$ operations.

²A function fulfills $f(x) \in \Omega(g(x))$ if there exist $N \in \mathbb{N}$ and $c \in \mathbb{R}$ such that $|f(n)| > c \cdot g(n)$ for all $n > N$.

Algorithm 2: Approximate Frontier Search.

Input: $\eta, L, x, \mathcal{P}, C_0$
Output: $p \in \mathcal{P}$ such that $L(p) = x$
 $e \leftarrow 1; t \leftarrow 1; n \leftarrow 1; \tau_1 \leftarrow 0;$
 $C \leftarrow \max\{\eta_1^2, C_0\}; M \leftarrow \lceil \log(C) \rceil^2;$
while true do
 $C \leftarrow 4 \cdot C; M \leftarrow M + 4;$
 for $i = 1, \dots, n$ **do**
 $m \leftarrow \lfloor C/\eta_i \rfloor - \tau_i;$
 run program p_i for m steps;
 if p_i halts and $L(p_i) = x$ **then return** $p_i;$
 $\tau_i \leftarrow \tau_i + m; t \leftarrow t + m;$
 end
 while true do
 $m \leftarrow \lfloor C/\eta_{n+1} \rfloor;$
 if $m < M$ **then break;**
 $n \leftarrow n + 1;$
 run program p_n for m steps;
 if p_n halts and $L(p_n) = x$ **then return** $p_n;$
 $\tau_n \leftarrow m; t \leftarrow t + m;$
 end
 $t_e \leftarrow t; e \leftarrow e + 1;$
end

Note that adding four to M and quadrupling C corresponds to maintaining the relation $M = \lceil \log(C) \rceil^2$.

Consider the number m computed in the for-loop. We show by induction that this number exceeds M : In the first iteration we have $n = 1$, such that the loop only runs over a single program, and $C = 4 \cdot \max\{\eta_1^2, C_0\}$ makes sure that $\lfloor C/\eta_1 \rfloor \geq M = \lceil \log(C) \rceil^2$ for suitable C_0 . In later iterations we know that $\lfloor C/\eta_i \rfloor - \tau_i \geq M$ was fulfilled in the previous iteration for all $i \in \{1, \dots, n\}$ (this is trivially fulfilled for the programs added in the inner while loop), implying $\lfloor C/\eta_i \rfloor \geq M$, which reads $\lfloor C/(4 \cdot \eta_i) \rfloor \geq M - 4$ in the notation of the current iteration. Together with $\tau_i \leq \lfloor C/(4 \cdot \eta_i) \rfloor$ and $\lfloor C/2 \rfloor > 4$ (for $C_0 \geq 1/2$) this implies $\lfloor C/\eta_i \rfloor \geq M$. Thus, all programs executed in an epoch are executed for at least M steps. Instead of choosing C_0 unnecessarily large, we may let the target complexity C start small and wait for C to exceed C_0 after finitely many epochs.

The budget available per epoch is linear in the number of program steps executed, which is $O(n \cdot M)$. It is easy to see that all additions, subtractions, and loop counters/comparisons easily fit into this time budget. The potentially most costly operation in the program is the division $\lfloor C/\eta_i \rfloor$. Its complexity is upper bounded by $O(\lceil \log(C) \rceil^2)$, which by construction coincides with $O(M)$. \square

Discussion

Frontier Search provides a very flexible alternative to Levin Search. This increased flexibility enables us to respect complicated complexity functions in the search.

The algorithm is known to work with a quite general set of complexity functions (see Proposition 7), while the still very flexible space of *proper* complexity functions is minimized by the algorithm exactly (see Proposition 10). For the more

restricted case of separable complexity functions we provide the algorithm Approximate Frontier Search which achieves constant overhead, while preserving the asymptotic properties of Frontier Search.

Even the relatively restricted case of separable complexity functions provides interesting search schemes. The only restriction on the growing sequence η_i of step costs is that it takes infinitely many different values. This excludes nearly everywhere constant priors, but it does not require the prior to be proper.

Non-separable cases may be of even greater interest. There are different reasons why we may wish to vary the cost of executing a command over time. On the one hand one may search for a program with runtime in the ‘right’ order of magnitude by only penalizing steps that exceed the next power of, e.g., ten. Or one may, like in example 14, increase the penalty over time, strongly favoring short execution time.

All these different search schemes can be realized with Frontier Search. The specification of a particular search scheme is implicitly done by providing a complexity function, which does not require any changes to the Frontier Search algorithm itself, and is intuitive and therefore easy to specify by the user.

Conclusion

We demonstrate the theoretically powerful search algorithm Frontier Search, which automatically finds programs optimal w.r.t. a given complexity criterion. It is provably more general than Levin Search and speed prior-based search in several respects: We can handle improper priors, and even time-varying execution costs under weak and intuitively meaningful technical conditions. For the case of separable complexity functions we propose Approximate Frontier Search, which achieves constant computational overhead.

Like Levin Search, the current approach is limited to programs computing a fixed output x . We leave generalizations to more relevant cases such as minimizing a loss function given data to future work.

Acknowledgments

This work was funded in part by SNF grants 200021-111968 and 200021-113364.

References

- L. A. Levin. Universal sequential search problems. *Problems of Information Transmission*, 9:265–266, 1973.
- M. Li and P. M. B. Vitányi. *An Introduction to Kolmogorov Complexity and its Applications*. 1993.
- J. Schmidhuber. The Speed Prior: a new simplicity measure yielding near-optimal computable predictions. In *Proceedings of the 15th Annual Conference on Computational Learning Theory (COLT 2002)*, pages 216–228. Springer, Sydney, Australia, 2002.
- T. Schaul and J. Schmidhuber. Towards Practical Universal Search. 2010. *Submitted to the Conference on Artificial General Intelligence*.

Artificial Scientists & Artists Based on the Formal Theory of Creativity

Jürgen Schmidhuber

IDSIA, Galleria 2, 6928 Manno-Lugano, Switzerland
University of Lugano & SUPSI, Switzerland

Abstract

I have argued that a simple but general formal theory of creativity explains many essential aspects of intelligence including science, art, music, humor. It is based on the concept of maximizing reward for the creation or discovery of *novel patterns* allowing for *improved* data compression or prediction. Here I discuss what kind of general bias towards algorithmic regularities we insert into our robots by implementing the principle, why that bias is good, and how the approach greatly generalizes the field of *active learning*. I emphasize the importance of limited computational resources for on-line prediction and compression, and provide discrete and continuous time formulations for ongoing work on building an Artificial General Intelligence (AGI) based on variants of the artificial creativity framework.

Introduction

Since 1990 I have built agents that may be viewed as simple artificial scientists or artists with an intrinsic desire to create / discover more *novel patterns*, that is, data predictable or compressible in hitherto unknown ways (Sch91b; Sch91a; SHS95; Sch02a; Sch06a; Sch07; Sch09c; Sch09b; Sch09a). The agents invent and conduct experiments to actively explore the world, always trying to learn new behaviors exhibiting previously unknown regularities. The agents embody approximations of a simple, but general, formal theory of creativity explaining essential aspects of human or non-human intelligence, including selective attention, science, art, music, humor (Sch06a; Sch07; Sch09c; Sch09b; Sch09a). Crucial ingredients are: (1) A predictor or compressor of the continually growing history of actions and sensory inputs, reflecting what's currently known about how the world works, (2) A learning algorithm that continually improves the predictor or compressor (detecting novel spatio-temporal patterns that subsequently become known patterns), (3) Intrinsic rewards measuring the predictor's or compressor's improvements due to the learning algorithm, (4) A reward optimizer or reinforcement learner, which translates those rewards into action sequences or behaviors expected to optimize future reward - the agent is intrinsically motivated to create additional novel

patterns predictable or compressible in previously unknown ways. We implemented the following variants: (A) Intrinsic reward as measured by improvement in mean squared prediction error (1991) (Sch91a), (B) Intrinsic reward as measured by relative entropies between the agent's priors and posteriors (1995) (SHS95), (C) Learning of probabilistic, hierarchical programs and skills through zero-sum intrinsic reward games of two players, each trying to out-predict or surprise the other, taking into account the computational costs of learning, and learning *when* to learn and *what* to learn (1997-2002) (Sch02a). (A, B, C) also showed experimentally how intrinsic rewards can substantially accelerate goal-directed learning and external reward intake. We also discussed (D) Mathematically optimal, intrinsically motivated systems driven by prediction progress or compression progress (2006-2009) (Sch06a; Sch07; Sch09c; Sch09b).

How does our formal theory of creativity and curiosity generalize the traditional field of **active learning**, e.g., (Fed72)? To optimize a function may require expensive data evaluations. Active learning typically just asks which data point to evaluate next to maximize information gain (1 step look-ahead), assuming all data point evaluations are equally costly. Our more general framework takes formally into account: (1) Agents embedded in an environment where there may be arbitrary delays between experimental actions and corresponding information gains, e.g., (SHS95; Sch91a), (2) The highly environment-dependent costs of obtaining or creating not just individual data points but data *sequences* of *a priori* unknown size, (3) Arbitrary algorithmic or statistical dependencies in sequences of actions & sensory inputs, e.g., (Sch02a; Sch06a), (4) The computational cost of learning new skills, e.g., (Sch02a). Unlike previous approaches, our systems measure and maximize algorithmic (Sol78; Kol65; LV97; Sch02b) novelty (learnable but previously unknown compressibility or predictability) of self-generated spatio-temporal patterns in the history of data and actions (Sch06a; Sch07; Sch09c; Sch09b).

How does the prediction progress drive / compression progress drive explain, say, **humor**? Consider the following statement: *Biological organisms are driven by*

the “Four Big F’s”: *Feeding, Fighting, Fleeing, Sexual Activity*. Some subjective observers who read this for the first time think it is funny. Why? As the eyes are sequentially scanning the text the brain receives a complex visual input stream. The latter is subjectively partially compressible as it relates to the observer’s previous knowledge about letters and words. That is, given the reader’s current knowledge and current compressor, the raw data can be encoded by fewer bits than required to store random data of the same size. But the punch line after the last comma is unexpected for those who expected another “F”. Initially this failed expectation results in sub-optimal data compression—storage of expected events does not cost anything, but deviations from predictions require extra bits to encode them. The compressor, however, does not stay the same forever: within a short time interval its learning algorithm improves its performance on the data seen so far, by discovering the non-random, non-arbitrary and therefore compressible pattern relating the punch line to previous text and previous knowledge about the “Four Big F’s.” This saves a few bits of storage. The number of saved bits (or a similar measure of learning progress) becomes the observer’s intrinsic reward, possibly strong enough to motivate him to read on in search for more reward through additional yet unknown patterns. The recent joke, however, will never be novel or funny again.

How does the theory informally explain the motivation to create or perceive **art and music** (Sch97b; Sch97a; Sch06a; Sch07; Sch09c; Sch09b; Sch09a)? For example, why are some melodies more interesting or aesthetically rewarding than others? Not the one the listener (composer) just heard (played) twenty times in a row. It became too subjectively predictable in the process. Nor the weird one with completely unfamiliar rhythm and tonality. It seems too irregular and contain too much arbitrariness and subjective noise. The observer (creator) of the data is interested in melodies that are unfamiliar enough to contain somewhat unexpected harmonies or beats etc., but familiar enough to allow for quickly recognizing the presence of a new learnable regularity or compressibility in the sound stream: a novel pattern! Sure, it will get boring over time, but not yet. All of this perfectly fits our principle: The current compressor of the observer or data creator tries to compress his history of acoustic and other inputs where possible. The action selector tries to find history-influencing actions such that the continually growing historic data allows for improving the compressor’s performance. The interesting or aesthetically rewarding musical and other subsequences are precisely those with previously unknown yet learnable types of regularities, because they lead to compressor improvements. The boring patterns are those that are either already perfectly known or arbitrary or random, or whose structure seems too hard to understand. Similar statements not only hold for other dynamic art including film and dance (take into account the compressibility of action sequences), but also for “static” art such as painting and sculpture, created

through action sequences of the artist, and perceived as dynamic spatio-temporal patterns through active attention shifts of the observer. When not occupied with optimizing *external* reward, artists and observers of art are just following their compression progress drive!

How does the theory explain the nature of **inductive sciences such as physics**? If the history of the entire universe were computable, and there is no evidence against this possibility (Sch06b), then its simplest explanation would be the shortest program that computes it. Unfortunately there is no general way of finding the shortest program computing any given data (LV97). Therefore physicists have traditionally proceeded incrementally, analyzing just a small aspect of the world at any given time, trying to find simple laws that allow for describing their limited observations better than the best previously known law, essentially trying to find a program that compresses the observed data better than the best previously known program. An unusually large compression breakthrough deserves the name *discovery*. For example, Newton’s law of gravity can be formulated as a short piece of code which allows for substantially compressing many observation sequences involving falling apples and other objects. Although its predictive power is limited—for example, it does not explain quantum fluctuations of apple atoms—it still allows for greatly reducing the number of bits required to encode the data stream, by assigning short codes to events that are predictable with high probability (Huf52) under the assumption that the law holds. Einstein’s general relativity theory yields additional compression progress as it compactly explains many previously unexplained deviations from Newton’s predictions. Most physicists believe there is still room for further advances, and this is what is driving them to invent new experiments unveiling novel, previously unpublished patterns (Sch09c; Sch09b; Sch09a). When not occupied with optimizing *external* reward, physicists are also just following their compression progress drive!

More Formally

Let us formally consider a learning agent whose single life consists of discrete cycles or time steps $t = 1, 2, \dots, T$. Its complete lifetime T may or may not be known in advance. In what follows, the value of any time-varying variable Q at time t ($1 \leq t \leq T$) will be denoted by $Q(t)$, the ordered sequence of values $Q(1), \dots, Q(t)$ by $Q(\leq t)$, and the (possibly empty) sequence $Q(1), \dots, Q(t-1)$ by $Q(< t)$. At any given t the agent receives a real-valued input $x(t)$ from the environment and executes a real-valued action $y(t)$ which may affect future inputs. At times $t < T$ its goal is to maximize future success or *utility*

$$u(t) = E_{\mu} \left[\sum_{\tau=t+1}^T r(\tau) \mid h(\leq t) \right], \quad (1)$$

where the reward $r(t)$ is a special real-valued input at time t , $h(t)$ the ordered triple $[x(t), y(t), r(t)]$ (hence $h(\leq t)$ is the known history up to t), and $E_\mu(\cdot \mid \cdot)$ denotes the conditional expectation operator with respect to some possibly unknown distribution μ from a set \mathcal{M} of possible distributions. Here \mathcal{M} reflects whatever is known about the possibly probabilistic reactions of the environment. For example, \mathcal{M} may contain all computable distributions (Sol78; LV97; Hut04). There is just one life, no need for predefined repeatable trials, no restriction to Markovian interfaces between sensors and environment, and the utility function implicitly takes into account the expected remaining lifespan $E_\mu(T \mid h(\leq t))$ and thus the possibility to extend the lifespan through appropriate actions (Sch05; Sch09d).

Recent work has led to the first reinforcement learning (RL) machines that are universal and optimal in various very general senses (Hut04; Sch02c; Sch09d). Such machines can in theory find out by themselves whether curiosity and creativity are useful or useless in a given environment, and learn to behave accordingly. In realistic settings, however, external rewards are extremely rare, and we cannot expect quick progress of this type, not even by optimal machines. But typically we can learn lots of useful behaviors even in absence of external rewards: unsupervised behaviors that just lead to predictable or compressible results and thus reflect the regularities in the environment, e. g., repeatable patterns in the world's reactions to certain action sequences. Here we argue again that a bias towards exploring previously unknown environmental regularities is *a priori* good in the real world as we know it, and should be inserted into practical AGIs, whose goal-directed learning will profit from this bias, in the sense that behaviors leading to external reward can often be quickly composed / derived from previously learnt, purely curiosity-driven behaviors. We shall not worry about the undeniable possibility that curiosity and creativity can actually be harmful and "kill the cat", that is, we assume the environment is benign enough. Based on our experience with the real world it may be argued that this assumption is realistic. Our explorative bias greatly facilitates the search for goal-directed behaviors in environments where the acquisition of external reward has indeed a lot to do with easily learnable environmental regularities.

To establish this bias, in the spirit of our previous work since 1990 (Sch91b; Sch91a; SHS95; Sch02a; Sch06a; Sch07; Sch09c; Sch09b; Sch09a) we simply split the reward signal $r(t)$ into two scalar real-valued components: $r(t) = g(r_{ext}(t), r_{int}(t))$, where g maps pairs of real values to real values, e.g., $g(a, b) = a + b$. Here $r_{ext}(t)$ denotes traditional *external* reward provided by the environment, such as negative reward in response to bumping against a wall, or positive reward in response to reaching some teacher-given goal state. The formal theory of creativity, however, is especially interested in $r_{int}(t)$, the internal or *intrinsic*

or *curiosity* or *creativity* or *aesthetic* reward, which is provided whenever the data compressor / internal world model of the agent improves in some measurable sense—for *purely creative* agents $r_{ext}(t) = 0$ for all valid t . The basic principle is essentially the one we published before in various variants (Sch91b; Sch91a; SHS95; Sch02a; Sch06a; Sch07; Sch09c; Sch09b; Sch09a):

Generate intrinsic curiosity reward or creativity reward for the controller in response to *improvements* of the predictor or history compressor.

This is a description of the agent's motivation - we conceptually separate the goal (finding or creating data that can be predicted / explained / compressed / understood better or faster than before) from the means of achieving the goal. Once the goal is formally specified in terms of an algorithm for computing curiosity rewards, let the controller's RL mechanism figure out how to translate such rewards into action sequences that allow the given compressor improvement algorithm to find and exploit previously unknown types of compressibility.

Computing Creativity Rewards

As pointed out above, predictors and compressors are closely related. Any type of partial predictability of the incoming sensory data stream can be exploited to improve the compressibility of the whole. We consider compressors that can deal with any prefix of the growing history, computing an output starting with $h(\leq t)$ for any time t ($1 \leq t < T$). (A compressor that wants to halt after t steps can easily be fixed / augmented by the trivial method that simply stores any raw additional data coming in after the halt.) Given some compressor program p able to compress history $h(\leq t)$, let $C(p, h(\leq t))$ denote p 's compression performance on $h(\leq t)$. One appropriate performance measure is

$$C_l(p, h(\leq t)) = l(p), \quad (2)$$

where $l(p)$ denotes the length of p , measured in number of bits: the shorter p , the more algorithmic regularity and compressibility and predictability and lawfulness in the observations so far. The ultimate limit for $C_l(p, h(\leq t))$ would be $K^*(h(\leq t))$, a variant of the Kolmogorov complexity of $h(\leq t)$, namely, the length of the shortest program (for the given hardware) that computes an output starting with $h(\leq t)$ (Sol78; Kol65; LV97; Sch02b).

$C_l(p, h(\leq t))$ does not take into account the time $\tau(p, h(\leq t))$ spent by p on computing $h(\leq t)$. In practical applications, however, time is essential. In fact, the predictor / compressor of the continually growing data typically will have to calculate its output online, that is, it will be able to use only a constant number of computational instructions per second to predict / compress new data. The goal of the possibly much slower learning algorithm must be to improve the compressor such that it keeps operating within those time limits, while compressing / predicting better than before.

A runtime-dependent performance measure inspired by concepts of optimal universal search (Lev73; Sch02c; Sch04; Sch06a; Sch09b) is

$$C_{lr}(p, h(\leq t)) = l(p) + \log \tau(p, h(\leq t)). \quad (3)$$

Here compression by one bit is worth as much as runtime reduction by a factor of $\frac{1}{2}$. From an asymptotic optimality-oriented point of view this is one of the best ways of trading off storage and computation time (Lev73; Sch02c; Sch04). In practice, however, we have mostly used online settings (one prediction per time step, and constant computational effort per prediction), and less universal adaptive compressors or predictors (Sch91b; Sch91a; SHS95; Sch02a; Sch06a).

So far we have discussed measures of compressor performance, but not of performance *improvement*, which is the essential issue in our creativity-oriented context. To repeat the point made above: *The important thing are the improvements of the compressor / predictor, not its compression performance per se.* Our creativity reward in response to the compressor's progress (due to some application-dependent compressor improvement algorithm) between times t and $t + 1$ is

$$r_{int}(t+1) = f[C(p(t), h(\leq t+1)), C(p(t+1), h(\leq t+1))], \quad (4)$$

where f maps pairs of real values to real values. Various alternative progress measures are possible; most obvious is $f(a, b) = a - b$. This corresponds to a discrete time version of maximizing the first derivative of subjective data compressibility. *Note that both the old and the new compressor have to be tested on the same data, namely, the history so far.* So compression progress between times t and $t + 1$ is defined based on the complexities of two programs that both compute $h(\leq t + 1)$, where the old one is trained only on $h(\leq t)$ and the new one also gets to see $h(t \leq t + 1)$. This is like $p(t)$ predicting data of time $t + 1$, then observing it, then learning something, then becoming a better predictor or compressor $p(t + 1)$.

Asynchronous Framework for Maximizing Creativity Reward

Compare (Sch06a; Sch07; Sch09b). Let $p(t)$ denote the agent's current compressor program at time t , $s(t)$ its current controller, and do:

Controller: At any time t ($1 \leq t < T$) do:

1. Let $s(t)$ use (parts of) history $h(\leq t)$ to select and execute $y(t + 1)$.
2. Observe $x(t + 1)$.
3. Check if there is non-zero creativity reward $r_{int}(t + 1)$ provided by the asynchronously running improvement algorithm of the compressor / predictor (see below). If not, set $r_{int}(t + 1) = 0$.
4. Let the controller's reinforcement learning (RL) algorithm use $h(\leq t + 1)$ including $r_{int}(t + 1)$ (and possibly also the latest available compressed version of

the observed data—see below) to obtain a new controller $s(t + 1)$, in line with objective (1). Note that some actions may actually trigger learning algorithms that compute changes of the compressor and the controller's policy, such as in (Sch02a). That is, the computational cost of learning can be taken into account by the reward optimizer, and the decision when and what to learn can be learnt as well (Sch02a).

Compressor / Predictor: Set p_{new} equal to the initial data compressor / predictor. Starting at time 1, repeat forever until interrupted by death at time T :

1. Set $p_{old} = p_{new}$; get current time step t and set $h_{old} = h(\leq t)$.
2. Evaluate p_{old} on h_{old} , to obtain performance measure $C(p_{old}, h_{old})$. This may take many time steps.
3. Let some (possibly application-dependent) compressor improvement algorithm (such as a learning algorithm for an adaptive neural network predictor, possibly triggered by a controller action) use h_{old} to obtain a hopefully better compressor p_{new} (such as a neural net with the same size and the same constant computational effort per prediction but improved predictive power and therefore improved compression performance (SH96)). Although this may take many time steps (and could be partially performed offline during "sleep"), p_{new} may not be optimal, due to limitations of the learning algorithm, e.g., local maxima.
4. Evaluate p_{new} on h_{old} , to obtain $C(p_{new}, h_{old})$. This may take many time steps.
5. Get current time step τ and generate creativity reward

$$r_{int}(\tau) = f[C(p_{old}, h_{old}), C(p_{new}, h_{old})], \quad (5)$$

e.g., $f(a, b) = a - b$.

This asynchronous scheme may cause long temporal delays between controller actions and corresponding creativity rewards, and may impose a heavy burden on the controller's RL algorithm whose task is to assign credit to past actions. (To inform the controller about beginnings of compressor evaluation processes etc., augment its input by unique representations of such events.) Nevertheless, there are RL algorithms for this purpose which are theoretically optimal in various senses (Sch06a; Sch07; Sch09c; Sch09b).

Continuous Time

In continuous time formulation, let $O(t)$ denote the state of subjective observer O at time t . The subjective simplicity or compressibility or regularity or beauty $B(D, O(t))$ of a sequence of observations and/or actions D is the negative number of bits required to encode D , given $O(t)$'s current limited prior knowledge and limited compression / prediction method. The observer-dependent and time-dependent subjective *Interestingness* or *Novelty* or *Surprise* or *Aesthetic Reward* or *Aesthetic Value* $I(D, O(t))$ is

$$I(D, O(t)) \sim \frac{\partial B(D, O(t))}{\partial t}, \quad (6)$$

the *first derivative* of subjective simplicity: as O improves its compression algorithm, formerly apparently random data parts become subjectively more regular and beautiful, requiring fewer and fewer bits for their encoding. Given its limited compression improver, at time t_0 the creativity goal of $O(t_0)$ is to select actions that will maximize

$$E \left[\int_{t=t_0}^T g[r_{int}(t), r_{ext}(t)] \partial t \right], \quad (7)$$

where E is an expectation operator (compare equation (1)); T is death; $r_{int}(t) = I(H(\leq t), O(t))$ is the momentary internal joy or intrinsic reward for compression progress through discovery of a novel pattern somewhere in $H(\leq t)$ (the history of actions and sensations until t); $r_{ext}(t)$ the current external reward if there is any; g is the function weighing external vs intrinsic rewards (e.g., $g(a, b) = a + b$). Note that there are at least two ways of getting intrinsic reward: execute a learning algorithm that improves the compression of the already known data (in online settings: without increasing computational needs of the compressor / predictor), or execute actions that generate more data, then learn to compress or understand the new data better.

Ongoing and Future Work

The systems described in the first publications on artificial curiosity and creativity (Sch91b; Sch91a; SHS95; Sch02a) already can be viewed as examples of implementations of a prediction / compression progress drive that encourages the discovery or creation of novel patterns, resulting in artificial scientists or artists with various types of computational limitations. To improve our previous implementations of the basic ingredients of the creativity framework (see introduction), and to build a continually growing, mostly unsupervised AGI, we will evaluate additional combinations of novel, advanced RL algorithms and adaptive compressors, and test them on humanoid robots such as the iCUB. That is, we will (A) study better practical adaptive compressors, in particular, recent, novel artificial recurrent neural networks (RNN) (HS97; SGG⁺09) and other general yet practically feasible methods for making predictions; (B) investigate under which conditions learning progress measures can be computed both accurately and efficiently, without frequent expensive compressor performance evaluations on the entire history so far; (C) study the applicability of recent improved RL techniques in the fields of artificial evolution, policy gradients, and others. In particular, recently there has been substantial progress in RL algorithms that are not quite as general as the universal ones (Hut04; Sch02c; Sch09d), but nevertheless capable of learning very general, program-like behavior. In particular, evolutionary methods can be used for training RNN, which are general computers. One especially effective family of methods uses cooperative coevolution to search the space of network components (*neurons* or individual

synapses) instead of complete networks. The components are *coevolved* by combining them into networks, and selecting those for reproduction that participated in the best performing networks (GSM08). Other recent promising RL techniques for RNN are based on the concept of policy gradients (SMSM99; SOR⁺08; WSPS08).

Conclusion and Outlook

In the real world external rewards are rare. But unsupervised AGIs using additional intrinsic rewards as described in this paper will be motivated to learn many useful behaviors even in absence of external rewards, behaviors that lead to predictable or compressible results and thus reflect regularities in the environment, such as repeatable patterns in the world's reactions to certain action sequences. Often a bias towards exploring previously unknown environmental regularities through artificial curiosity / creativity is *a priori* desirable because goal-directed learning may greatly profit from it, as behaviors leading to external reward may often be rather easy to compose from previously learnt curiosity-driven behaviors. It may be possible to formally quantify this bias towards novel patterns in form of a mixture-based prior (Sol78; LV97; Sch02c; Hut04), a weighted sum of probability distributions on sequences of actions and resulting inputs, and derive precise conditions for improved expected external reward intake. Intrinsic reward may be viewed as analogous to a *regularizer* in supervised learning, where the prior distribution on possible hypotheses greatly influences the most probable interpretation of the data in a Bayesian framework (Bis95) (for example, the well-known weight decay term of neural networks is a consequence of a Gaussian prior with zero mean for each weight). Following the introductory discussion, some of the AGIs based on the creativity principle will become scientists, artists, or comedians.

References

- C. M. Bishop. *Neural networks for pattern recognition*. Oxford University Press, 1995.
- V. V. Fedorov. *Theory of optimal experiments*. Academic Press, 1972.
- F. J. Gomez, J. Schmidhuber, and R. Miikkulainen. Efficient non-linear control through neuroevolution. *Journal of Machine Learning Research JMLR*, 9:937–965, 2008.
- S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.
- D. A. Huffman. A method for construction of minimum-redundancy codes. *Proceedings IRE*, 40:1098–1101, 1952.
- M. Hutter. *Universal Artificial Intelligence: Sequential Decisions based on Algorithmic Probability*. Springer, Berlin, 2004. (On J. Schmidhuber's SNF grant 20-61847).

- A. N. Kolmogorov. Three approaches to the quantitative definition of information. *Problems of Information Transmission*, 1:1–11, 1965.
- L. A. Levin. Universal sequential search problems. *Problems of Information Transmission*, 9(3):265–266, 1973.
- M. Li and P. M. B. Vitányi. *An Introduction to Kolmogorov Complexity and its Applications (2nd edition)*. Springer, 1997.
- J. Schmidhuber. Curious model-building control systems. In *Proceedings of the International Joint Conference on Neural Networks, Singapore*, volume 2, pages 1458–1463. IEEE press, 1991.
- J. Schmidhuber. A possibility for implementing curiosity and boredom in model-building neural controllers. In J. A. Meyer and S. W. Wilson, editors, *Proc. of the International Conference on Simulation of Adaptive Behavior: From Animals to Animats*, pages 222–227. MIT Press/Bradford Books, 1991.
- J. Schmidhuber. Femmes fractales, 1997.
- J. Schmidhuber. Low-complexity art. *Leonardo, Journal of the International Society for the Arts, Sciences, and Technology*, 30(2):97–103, 1997.
- J. Schmidhuber. Exploring the predictable. In A. Ghosh and S. Tsutsui, editors, *Advances in Evolutionary Computing*, pages 579–612. Springer, 2002.
- J. Schmidhuber. Hierarchies of generalized Kolmogorov complexities and nonenumerable universal measures computable in the limit. *International Journal of Foundations of Computer Science*, 13(4):587–612, 2002.
- J. Schmidhuber. The Speed Prior: a new simplicity measure yielding near-optimal computable predictions. In J. Kivinen and R. H. Sloan, editors, *Proceedings of the 15th Annual Conference on Computational Learning Theory (COLT 2002)*, Lecture Notes in Artificial Intelligence, pages 216–228. Springer, Sydney, Australia, 2002.
- J. Schmidhuber. Optimal ordered problem solver. *Machine Learning*, 54:211–254, 2004.
- J. Schmidhuber. Completely self-referential optimal reinforcement learners. In W. Duch, J. Kacprzyk, E. Oja, and S. Zadrozny, editors, *Artificial Neural Networks: Biological Inspirations - ICANN 2005, LNCS 3697*, pages 223–233. Springer-Verlag Berlin Heidelberg, 2005. Plenary talk.
- J. Schmidhuber. Developmental robotics, optimal artificial curiosity, creativity, music, and the fine arts. *Connection Science*, 18(2):173–187, 2006.
- J. Schmidhuber. Randomness in physics. *Nature*, 439(3):392, 2006. Correspondence.
- J. Schmidhuber. Simple algorithmic principles of discovery, subjective beauty, selective attention, curiosity & creativity. In *Proc. 10th Intl. Conf. on Discovery Science (DS 2007)*, *LNAI 4755*, pages 26–38. Springer, 2007. Joint invited lecture for *ALT 2007 and DS 2007*, Sendai, Japan, 2007.
- J. Schmidhuber. Art & science as by-products of the search for novel patterns, or data compressible in unknown yet learnable ways. In M. Botta, editor, *Multiple ways to design research. Research cases that reshape the design discipline*, Swiss Design Network - Et al. Edizioni, pages 98–112. Springer, 2009.
- J. Schmidhuber. Driven by compression progress: A simple principle explains essential aspects of subjective beauty, novelty, surprise, interestingness, attention, curiosity, creativity, art, science, music, jokes. In G. Pezzulo, M. V. Butz, O. Sigaud, and G. Baldassarre, editors, *Anticipatory Behavior in Adaptive Learning Systems. From Psychological Theories to Artificial Cognitive Systems*, volume 5499 of *LNCS*, pages 48–76. Springer, 2009.
- J. Schmidhuber. Simple algorithmic theory of subjective beauty, novelty, surprise, interestingness, attention, curiosity, creativity, art, science, music, jokes. *SICE Journal of the Society of Instrument and Control Engineers*, 48(1):21–32, 2009.
- J. Schmidhuber. Ultimate cognition à la Gödel. *Cognitive Computation*, 1(2):177–193, 2009.
- J. Schmidhuber, A. Graves, F. J. Gomez, S. Fernandez, and S. Hochreiter. *How to Learn Programs with Artificial Recurrent Neural Networks*. Invited by Cambridge University Press, 2009. In preparation.
- J. Schmidhuber and S. Heil. Sequential neural text compression. *IEEE Transactions on Neural Networks*, 7(1):142–146, 1996.
- J. Storck, S. Hochreiter, and J. Schmidhuber. Reinforcement driven information acquisition in non-deterministic environments. In *Proceedings of the International Conference on Artificial Neural Networks, Paris*, volume 2, pages 159–164. EC2 & Cie, 1995.
- R. S. Sutton, D. A. McAllester, S. P. Singh, and Y. Mansour. Policy gradient methods for reinforcement learning with function approximation. In S. A. Solla, T. K. Leen, and K.-R. Müller, editors, *Advances in Neural Information Processing Systems 12, [NIPS Conference, Denver, Colorado, USA, November 29 - December 4, 1999]*, pages 1057–1063. The MIT Press, 1999.
- R. J. Solomonoff. Complexity-based induction systems. *IEEE Transactions on Information Theory*, IT-24(5):422–432, 1978.
- F. Sehnke, C. Osendorfer, T. Rückstieß, A. Graves, J. Peters, and J. Schmidhuber. Policy gradients with parameter-based exploration for control. In *Proceedings of the International Conference on Artificial Neural Networks ICANN*, 2008.
- D. Wierstra, T. Schaul, J. Peters, and J. Schmidhuber. Fitness expectation maximization. In *Proceedings of Parallel Problem Solving from Nature (PPSN 2008)*, 2008.

Quantitative Spatial Reasoning for General Intelligence

Unmesh Kurup and Nicholas L. Cassimatis

Rensselaer Polytechnic Institute
110 8th St, Troy, NY 12180
{kurup,cassin}@rpi.edu

Abstract

One of the basic requirements of an intelligent agent is the ability to represent and reason about space. While there are a number of approaches for achieving this goal, the recent gains in efficiency of the Satisfiability approach have made it a popular choice. Modern propositional SAT solvers are efficient for a wide variety of problems. However, conversion to propositional SAT can sometimes result in a large number of variables and/or clauses. Diagrams represent space as collections of points (regions) while preserving their overall geometric character. This representation allows reasoning to be performed over (far fewer number of) regions instead of individual points. In this paper, we show how the standard DPLL algorithm augmented with diagrammatic reasoning can be used to make SAT more efficient when reasoning about space. We present DPLL-S, a complete SAT solver that utilizes diagrammatic representations when reasoning about space, and evaluate its performance against other SAT solvers.

Introduction

One of the fundamental aspects of general intelligence is the ability to represent and reason about space. Successful approaches such as the Region Connection Calculus (Randell, Cui, and Cohn 1992) have concentrated on qualitative spatial reasoning but an intelligent agent must be capable of reasoning about quantitative (or metric) space as well. With the advent of faster and more efficient SAT solvers, reasoning via translation to SAT has yielded results that are often as good as or even better than standard approaches. However, one drawback of propositionalizing first-order theories is the inherent explosion in variables and clauses. This is particularly true in quantitative spatial reasoning where space is usually represented using a simple Cartesian system. An object in this representation has a (x, y) coordinate that uniquely identifies where it lies in the space. This coordinate is also used to reason about the spatial relationships that the object forms with other objects in the same space. Translating a problem in this metric domain into propositional SAT involves propositions that capture the relationships between each pair of points in the space as well as instantiating propositions that capture the possibility that an object can be located anywhere in the space. As the space grows larger, the number of variables and clauses in the translation increase, making it more difficult to solve the problem.

The inclusion of domain-specific knowledge into the satisfiability process is captured under the umbrella of Satisfiability-Modulo Theories or SMT (DeMoura and Rue 2002). In SMT, parts of the formula that refer to the specific theory are handed over to the theory-specific solver while the rest is handled by the SAT solver. Quantifier Free Integer Difference Logic (QF-IDL) is one of the theories commonly used for reasoning about space in the SMT approach. In QF-IDL, spatial relationships between objects are represented as a set of inequalities. For example, the inequality $a_x \leq b_x - 1$, where a_x and b_x are the x-coordinates of objects a and b respectively, represents the fact that object a is to the left of object b . The inequalities can be represented as a graph structure and efficient algorithms exist that can check for satisfiability by checking for the presence of loops in the graph (Cotton 2005). However, while these inequalities are efficient in capturing the relationship between point objects, expanding their use to capture the representation of 2-d shapes has at least two drawbacks - One, the number of inequalities needed to represent a shape increases as the complexity of the shape increases since a shape is represented as a set of inequalities between its vertices. Two, when a shape (even a simple one such as rectangle) is allowed to rotate, the relationship between its vertices change and inequalities will have to be written for each possible rotation of the shape. The number of such sets of inequalities depends on the fineness to which the rotation needs to be captured. In this paper, we propose the use of diagrammatic models as the appropriate theory for representing and reasoning about space and show how the SAT approach can be augmented to use diagrammatic models as appropriate during solving. We evaluate our approach against the MiniMaxSat algorithm (Heras, Larrosa, and Oliveras 2008) in a spatial constraint satisfaction problem.

Satisfiability with Spatial Reasoning

Consider a 3x3 grid. To encode the information that object a is *Next* to object b , we would need a SAT formula like the following (in non-CNF form): $(AT(a, 1, 1) \wedge (AT(b, 1, 2) \vee AT(b, 2, 1) \vee AT(b, 2, 2))) \vee (AT(a, 1, 2) \wedge (AT(b, 1, 1) \vee AT(b, 2, 1) \vee AT(b, 2, 2) \vee AT(b, 1, 3) \vee AT(b, 2, 3))) \vee \dots$ and so on till every location in the grid has been accounted for. Even for simple spatial relations such as *Left* or *Above* the number of variables and clauses needed will grow as the

size of the grid grows. Propositionalizing space for the purposes of SAT is, thus, an expensive approach. Diagrams, on the other hand, can represent information more compactly by abstracting individual locations that share constraints into groups. We extend the current satisfiability approach to include diagrammatic models as part of the representation. We first introduce the concept of a diagram.

Diagram

Definition 2.1 (Object Location) Given a grid of size N_g and an object a , we define

- $L(a) = (x, y) | 1 \leq x, y \leq N_g$ where (x, y) is the location of object a in the grid.
- $L_x(a) = x | 1 \leq x \leq N_g$ and $L_y(a) = y | 1 \leq y \leq N_g$ where x and y are the x - and y -coordinates of a in the grid

Definition 2.2 (Relations) Spatial reasoning is based on the constraints that hold between objects in the space. We define five spatial constraint types ($T = \{Left|Right|Above|Below|Near\}$) that we use in this paper. More can be defined and added as necessary.

Given a grid of size N_g , objects a, b and N_e a nearness value, a constraint c holds between objects a and b iff one of the following hold

- $c = Left$ and $L_x(a) < L_x(b)$
- $c = Right$ and $L_x(a) > L_x(b)$
- $c = Above$ and $L_y(a) < L_y(b)$
- $c = Below$ and $L_y(a) > L_y(b)$
- $c = Near$ and $L_x(b) - N_e \leq L_x(a) \leq L_x(b) + N_e, L_y(b) - N_e \leq L_y(a) \leq L_y(b) + N_e$

Definition 2.3 (Possibility Space) As mentioned earlier, one of the disadvantages of propositionalizing space is the need to account for the possibility of an object being in every location in the space. However, in qualitative reasoning, it is the relationships that hold between objects that matter rather than their exact locations in space. For example, in satisfying the constraint $Left(b)$ for an object a , we don't care whether a is one spot to the left of b or two spots to the left and so on. This means that we can generalize away from exact locations to location groups where the members of each group share certain common spatial constraints. Generalization in this manner leads to lesser number of individuals resulting in better performance when converted to propositional SAT. The concept of the possibility space (Wintermute and Laird 2007) allows us to do this generalization. A possibility space is a set of points that satisfy some set of spatial constraints. Every object in a diagram resides in a possibility space and spatial relationships between objects can be computed by finding intersections between these possibility spaces. For example, given a 3x3 grid, an object a at location (2,2), and an object b with constraints $C(b) = \{Left(a), Above(a)\}$, $P_s(Left(a)) = \{(1, 1), (1, 2), (1, 3)\}$, $P_s(Above(a)) = \{(1, 1), (2, 1), (3, 1)\}$ and $P_s(b) = (P_s(Left(a)) \cap P_s(Above(a))) = \{(1, 1)\}$.

We define possibility spaces as follows: Given a grid of side N_g , an object a and $c \in T$, we define

1. $P_s(c(a), I_i)$, the possibility space of a spatial constraint $c(a)$ with truth value I_i as follows
 - $c = Left, I_i = true, P_s(c(a), I_i) = \{(x_i, y_i) | 1 \leq x_i, y_i \leq N_g; x_i < L_x(a)\}$
 - $c = Left, I_i = false, P_s(c(a), I_i) = \{(x_i, y_i) | 1 \leq x_i, y_i \leq N_g; x_i \geq L_x(a)\}$
 - similarly for $c = Right, Above, Below$ and $I_i = true, false$
 - $c = Near, I_i = true, P_s(c(a), I_i) = \{(x_i, y_i) | 1 \leq x_i, y_i \leq N_g; L_x(a) - N_e \leq x_i \leq L_x(a) + N_e; L_y(a) - N_e \leq y_i \leq L_y(a) + N_e; 1 \leq N_e \leq N_g\}$
 - $c = Near, I_i = false, P_s(c(a), I_i) = \{(x_i, y_i) | 1 \leq x_i, y_i \leq N_g; (x_i, y_i) \notin P_s(Near(a), true)\}$
2. $P_{s1} \cap P_{s2}$, the intersection of two possibility spaces as follows
 - $P_{s1} \cap P_{s2} = \{(x_i, y_i) | (x_i, y_i) \in P_{s1}, (x_i, y_i) \in P_{s2}\}$
3. $P_s(a)$, the possibility space of an object a as follows
 - $P_s(a) = \bigcap_{i=1}^{|C(a)|} P_s(c_i(a), I(c_i(a)))$ where $C(a) = \{c_1, \dots, c_k\}$ is the set of spatial constraints on a and $I(c_i(a)) = true|false$ is the truth value of the constraint $c_i(a)$

Definition 2.4 (Diagram) Finally, object locations, possibility spaces and relations come together in the diagrammatic representation. A diagram is a collection of objects with their locations and possibility spaces. Reasoning about the spatial relationships between objects in a diagram can be accomplished using the objects' possibility spaces and locations. Formally, we define a diagram as follows:

A diagram d is a 6-tuple $\langle N_d, O, T, C, I, L \rangle$ where

- N_d denotes the side of the diagram (for the purposes of this paper, diagrams are considered to be square)
- $O = \{a_1, a_2, \dots, a_k\}$ is a set of objects
- $T = \{Left|Right|Above|Below|Near\}$ is a set of relation types
- C is a set of spatial constraints from T that holds between objects in O
- $I : C \rightarrow true|false$ is an assignment of truth values to the constraints in C
- $L : O \rightarrow N_d \times N_d$, the location of objects in the diagram

C is a set of spatial constraints for objects in the diagram. If $C(a_i) = \emptyset$, then a_i 's possibility space is the entire diagram. $L(a_i)$ is the location of a_i in the diagram such that $L(a_i) \in P_s(a_i)$. This location is chosen such that it is at the center of the object's possibility space.

Definition 2.5 A diagram d satisfies a set of spatial constraints C iff for each $c(a, b) \in C$ and $a, b \in O$, the constraint C holds between a and b in d .

Satisfiability with Spatial Reasoning (SAT-S)

In order to combine the best of both SAT and diagrammatic reasoning, we introduce a version of SAT called SAT-S, that

allows for the representation of spatial information using diagrammatic models. Formally,

A problem specification in SAT-S is given by the 6-tuple $S_s = \langle \phi, P, O, T, C, M \rangle$ where

- ϕ is a SAT formula in CNF form
- P is the set of variables in ϕ
- O is a set of objects
- $T = \{Left|Right|Above|Below|Near\}$ is a set of spatial relation types
- C is a set of constraints of the form $c(a, b)$ where $c \in T$ and $a, b \in O$
- $M : P \rightarrow C$ is a mapping from P to C

ϕ and P are the same as in SAT. O is a set of objects in the domain. T is the set of spatial relation types that are relevant in the domain. C is a set of constraints from the set $T \times O \times O$ and represents the spatial relations that are important in the domain. Finally, M is a mapping from the set of variables to the set of relations. This mapping allows the SAT-S solver to recognize variables that represent spatial literals. For convenience, in the remainder of this paper, we refer to such variables as spatial variables, even though they are first-order propositions.

Solutions in SAT-S

A solution (model) to a problem in SAT-S is an assignment of truth values, $I_p = \{true, false, neutral\}$ to the variables in P . A valid model in SAT-S is an assignment I_p such that

- Every clause in the ϕ evaluates to true and
- $\{\exists P' \subseteq P | (\forall p \in P') I_p(p) \neq neutral \wedge M(p) \neq \emptyset\} \Rightarrow |D| > 0 \wedge (\forall d \in D, p \in P') d \text{ satisfies } M(p)$, where D is a set of diagrams constructed during the solving process.

The set of diagrams D represents the possible configurations of objects in O given the spatial variables that have been assigned values. If a problem in SAT-S has propositions that denote spatial relations and these propositions have been assigned values in the solution then there must be at least one diagram in D and every diagram in D must satisfy these spatial relations.

DPLL-S - An Algorithm for Solving Problems in SAT-S

Algorithm 1 shows the DPLL-S algorithm. The main difference from the original DPLL algorithm is as follows: When a value is set for a variable, DPLL-S checks to see if it is a spatial variable. If it is not, the algorithm proceeds like in standard DPLL. If it is a spatial variable, say $M(v) = c(a, b)$, DPLL-S constructs a set of diagrams that satisfy the proposition in the following way –

1. If there are no diagrams (this is the first spatial proposition whose value has been set), it creates an empty diagram d and adds b (assigns its possibility space and a location within the possibility space) to this diagram.

Algorithm 1 DPLL-S

```

Procedure DPLL-S( $C, P, D, M$ )
  if  $C$  is empty, return satisfiable
  if  $C$  contains an empty clause,
    return unsatisfiable
  if  $C$  contains a unit clause
    if variable in unit clause is spatial
       $D' = \text{Propagate}(D, M(v), true)$ ;
      if  $D'$  is empty, return unsatisfiable
      return DPLL-S( $C(v=true), P, D', M$ )
    return DPLL-S( $C(v=true), P, D, M$ )
  pick an unassigned variable  $v$ 
  if  $v$  is spatial
     $D' = \text{Propagate}(D, M(v), true)$ 
    if  $D$  is empty, return unsatisfiable
  else
     $D' = D$ 
    if DPLL-S( $C(v=true), P, D', M$ ) is satisfiable,
      return satisfiable
    else
      if  $v$  is spatial
         $D' = \text{Propagate}(D, M(v), false)$ 
        if  $D$  is empty, return unsatisfiable
      else
         $D' = D$ 
      return DPLL-S( $C(v=false), P, D', M$ )

```

2. If one of the objects is present in the diagrams in D , then from each diagram d_i in D , new diagrams $\{d_{i1}, \dots, d_{ik}\}$ are constructed such that every d_{ij} satisfies $c(a, b)$. In addition, each d_{ij} also satisfies a different subset of the set of all possible relevant relations between a and O given d_i where O is the set of all objects in d . Thus, the new set of diagrams $D' = \{d_{i1} \cup \dots \cup d_{ik}\}$ where $|D| = l$. If $D' = \emptyset$, the procedure returns unsatisfiable.
3. If both objects are present in D , $D' = \{d_i | d \in D; d_i \text{ satisfies } c(a, b)\}$. If $D' = \emptyset$, the procedure returns unsatisfiable.

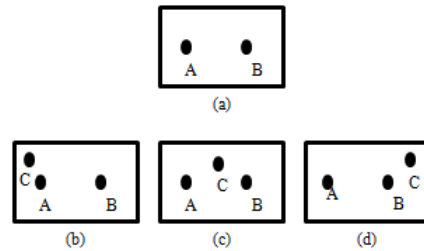


Figure 1: Diagrams created by Propagate for Above(c,a)

The creation of new diagrams once a spatial variable is found is done by the propagate method (shown in Algorithm 2) which takes a spatial constraint $M(v)$, the truth value I_v of this constraint and a set of diagrams D that satisfy a set of constraints C and produces a new set of diagrams D' such that every diagram in D' satisfies $C \cup M(v)$ (if $I_v = true$) or $C \cup \neg M(v)$ (if $I_v = false$). Each individual diagram in D' is constructed such that it captures some subset of the

Algorithm 2 Propagate

```

Procedure Propagate( $D, c(a, b), tvalue$ )
  if  $a$  and  $b$  in  $d \in D$ ,
    eliminate all  $d$  from  $D$ 
    that do not satisfy  $c(a, b)$ .
  return  $D$ 
if  $D$  is empty,
  create a new diagram  $d$ ,
  add  $b$  to  $d$ 
 $D' =$  empty set of diagrams
for every  $d \in D$ 
   $C = \{c(a, o) | c \in T; o \in O\}$ 
   $I$  is the set of all
    possible truth value assignments
    to  $c \in C$  and  $I_i(c_j)$  is the  $i$ th
    truth value assignment for  $c_j$ .
   $D' = D \cup d'_i$  where
   $d'_i = d_i \cup a$  where
   $P_s(a) =$ 
   $P_s(r(b)) \cap \left( \bigcap_{j=1}^{|C|} P_s(c_j, I_i(c_j)) \right), P_s(a) \neq \emptyset$ 

```

set of all possible relevant spatial relationships between the new object being introduced into the diagram and the objects currently in the diagram. Together, these diagrams capture all possible spatial relationships that the new object can be in given the constraint set $C \cup M(v)$.

As an example, consider the diagram in Fig 1(a). It has two objects A and B with $C(A) = Left(B)$. For expository purposes, let us assume that there are only two types of spatial relations that are relevant to our problem – *Left* and *Above*, i.e., $T = \{Left, Above\}$. Let $M(v) = Above(c, a)$ where c is the new object being introduced into the diagram. Then,

$$\begin{aligned}
 d_1 &= d + c \text{ where } P(c) = P_s(Above(A) \cap \neg Left(A) \cap \\
 &\quad \neg Left(B) \cap \neg Above(b)) \\
 &\vdots \\
 d_8 &= d + c \text{ where } P_s(c) = P_s(Above(A) \cap Left(A) \cap \\
 &\quad Left(B) \cap Above(b))
 \end{aligned}$$

In general, given a diagram d , a new object c , a set of constraints C and an assignment of truth values I to elements in C ,

$$\begin{aligned}
 D' &= D + d_i, \text{ where} \\
 d_i &= d + c \text{ where } P_s(c) = P_s(r(b)) \cap \\
 &\quad \left(\bigcap_{j=1}^{|C|} P_s(c_j, I(c_j)) \right), P_s(c) \neq \emptyset
 \end{aligned}$$

In the worst case, there are $|D| \times 2^{|T| \times |C| - 1}$ diagrams produced at each propagate step. On average, only a subset of these diagrams are possible. In the previous example, the diagrams where $P_s(c) = P_s(Above(A) \cap Left(A) \cap \neg Left(B) \cap Above(B))$ is not possible because an object cannot be both to the *Left* of A and not to the *Left* of B

given $C(a) = \{Left(b)\}$. In such a case, $P_s(c) = \emptyset$ and the diagram is eliminated.

Completeness

Lemma 3.1 - By definition of *Left*, *Right*, *Above*, *Below* and *Near*, the following are true

1. $Left(a, b) = Right(b, a)$ and vice versa
2. $Above(a, b) = Below(b, a)$ and vice versa
3. $Near(a, b) = Near(b, a)$

Lemma 3.2 - Given a diagram of size N_d , set of spatial constraints $C(b) = \{c_1(a_1), \dots, c_j(a_k)\}$ on object b , there is a location $L(b)$ for object b that satisfies $C(b)$ iff $\left(\bigcap_{j=1}^{|C(b)|} P_s(c_j, I(c_j)) \right) \neq \emptyset$

Proof: By definition of L in diagrams.

Theorem 3.1 Given a problem in SAT-S, $S_s = \langle \phi, P, O, T, C, I, M \rangle$ for which a solution $\langle I'D \rangle$ exists, where $|D| > 0$, DPLL-S will find at least one diagram d such that d satisfies the constraints C

Proof: In each execution step DPLL-S handles one of the following cases

1. Not a spatial variable. Continue as in DPLL
2. $|C| = k + 1$ and $|O| = m + 1$. A new object a_{m+1} is added to D with the constraint $c_{k+1}(a_j)$ where $a_j \in O$ is an object in D . Then, by construction and by lemma 3.2, all possible relationships between a_{m+1} and all objects in D including a_j are satisfied by D' .
3. $|C| = k + 1$ and $|O| = m$. A new constraint $c_{k+1}(a_j)$ is added to an existing object a_i .
 - (a) a_i was added after a_j - By construction and lemma 3.2, all possible relationships between a_i and a_j given C are satisfied.
 - (b) a_j was added after a_i - By construction, lemma 3.2 and lemma 3.1, all possible relationships between a_i and a_j given C are satisfied.

Hence proved.

Experiments

We compared the performance of DPLL-S to that of MiniMaxSAT (Heras, Larrosa, and Oliveras 2008) in a spatial reasoning problem called the placement problem, a generic version of the 8-queens puzzle. The choice of MiniMaxSAT as the standard for a traditional SAT solver was one of convenience. zChaff or SATzilla may have been marginally better but the reasons why MiniMaxSAT performs so poorly holds for the other solvers as well.

The Placement Problem

The objective of the Placement problem is to locate a set of objects in a grid space such that they satisfy a set of spatial relations. Formally, a placement problem P can be defined as a 4-tuple $P = \langle N_g, O, T, R \rangle$ where N_g is the side of a square grid, O is a set of objects, $T = \{Left|Right|Above|Below|Near\}$ is a set of spatial relation types, $N_e = N_g/3$ is a nearness value and R is a set

	SAT						SAT-S					
	#Near=1		#Near=2		#Near=3		#Near=1		#Near=2		#Near=3	
Grid Size	#vars	#clauses	#vars	#clauses	#vars	#clauses	#vars	#clauses	#vars	#clauses	#vars	#clauses
25	153	128893	153	161952	153	377511	3	3	3	3	3	3
35	213	461408	213	765482	213	1217816	3	3	3	3	3	3
45	273	770949	273	2103939	273	2282724	3	3	3	3	3	3
55	333	1795243	333	5752032	333	5340666	3	3	3	3	3	3
65	393	3609053	393	10913087	393	16353831	3	3	3	3	3	3
75	453	10119709	453	16119434	453	22119159	3	3	3	3	3	3
85	513	10239193	513	20424402	513	36685921	3	3	3	3	3	3

Table 1: Number of variables and clauses for SAT and SAT-S

Grid Size	MiniMaxSAT times			DPLL-S times			MiniMaxSAT(SAT-PP) times		
	#Near=1	#Near=2	#Near=3	#Near=1	#Near=2	#Near=3	#Near=1	#Near=2	#Near=3
25	0.34s	0.43s	1.31s	0.016s	0.0s	0.0s	0.82s	0.25s	0.02s
35	1.590s	2.46s	3.87s	0.0s	0.016s	0.0s	0.06s	0.02s	0.63s
45	2.334s	6.61s	6.93s	0.0s	0.0s	0.0s	0.5s	0.27s	0.06s
55	5.36s	17.46s	15.83s	0.0s	0.0s	0.0s	1.13s	0.18s	0.19s
65	10.97s	1m22.7s	2m26.4s	0.0s	0.0s	0.0s	1.09s	0.17s	.07s
75	1m54.5s	2m31.6s	10m39s	0.0s	0.0s	0.0s	0.23s	0.26s	0.32s
85	1m59.6s	5m31.2s	-	0.0s	0.0s	0.0s	0.36s	0.03s	0.21s
90	9m38.8s	-	-	0.0s	0.0s	0.0s	5.22s	0.03s	0.25s

Table 2: Comparison of MiniMaxSat and DPLL-S on Placement problems of increasing grid size

of spatial relations that hold between objects in O . A solution to the placement problem P is given by an assignment $S : O \rightarrow N_g \times N_g$ that identifies a unique location for each object $a \in O$ such that all constraints in R are satisfied.

Translation from P to SAT

Literals For every object $a_i \in O$ and location (k, l) in the grid, we assign two variables of the form $AT_{x_k}(a_i)$ and $AT_{y_l}(a_i)$, corresponding to the object's x and y-coordinates. A pair of variables $AT_{x_k}(a_i)$ and $AT_{y_l}(a_i)$ are true iff the object a_i is at location (k, l) in the grid. For every relation $r(a_i, a_j)$ in R , we add a variable r_m such that r_m is true iff $r(a_i, a_j) \in R$.

Clauses For every object a_i , we add two clauses $((AT_{x_1}(a_i)) \vee \dots \vee AT_{x_{N_g}}(a_i))$ and $((AT_{y_1}(a_i)) \vee \dots \vee AT_{y_{N_g}}(a_i))$

For each *Left* relation $Left(a_i, a_j)$, we add clauses that capture the following constraint $Left(a_i, a_j) \Rightarrow [((AT_{x_1}(a_i) \wedge AT_{x_2}(a_j)) \vee \dots \vee (AT_{x_1}(a_i) \wedge AT_{x_{N_g}}(a_j))) \vee \dots \vee ((AT_{x_{N_g-1}}(a_i) \wedge AT_{x_{N_g}}(a_j)))]$

We add similar constraints for each *Above*, *Right*, and *Below* relations.

For each *Near* relation $Near(a_i, a_j)$, we add clauses that capture the following constraint $Near(a_i, a_j) \Rightarrow \bigvee_{k=1, l=1}^{N_g, N_g} (AT_{x_k}(a_i) \wedge AT_{y_l}(a_i) \wedge (\bigvee_{m=k-N_e, n=l-N_e}^{k+N_e, l+N_e} (AT_{x_m}(a_j) \wedge AT_{y_n}(a_j))))$ where $1 \leq m, n \leq N_g$

For every object a_i , we add clauses that capture the constraint that an object can be only at one location in the grid $[(AT_{x_1}(a_i) \wedge AT_{y_1}(a_i) \wedge \neg(AT_{x_1}(a_i) \wedge AT_{y_2}(a_i)) \wedge \dots \wedge \neg(AT_{x_{N_g}}(a_i) \wedge AT_{y_{N_g}}(a_i))) \vee \dots \vee (AT_{x_{N_g}}(a_i) \wedge AT_{y_{N_g}}(a_i))]$

For every location (k, l) , we add clauses that capture the constraint that there can be only object at that location $[((A(T_{x_k}(a_1) \wedge AT_{y_l}(a_1)) \wedge \neg(A(T_{x_k}(a_2) \wedge AT_{y_l}(a_2)) \wedge \dots \wedge \neg(A(T_{x_k}(a_{|O|}) \wedge AT_{y_l}(a_{|O|})))) \vee \dots \vee (A(T_{x_k}(a_{|O|}) \wedge AT_{y_l}(a_{|O|})))]$

All constraints were converted to CNF form without an exponential increase in the number of clauses. Due to space constraints, we do not provide a proof of this translation but a proof by contradiction is straightforward.

Translation from P to SAT-S

Literals For every relation $r(a_i, a_j)$ in R , we add a variable r_m , and set $M(r_m) = r(a_i, a_j)$

Clauses For every relation $r(a_i, a_j)$ that is true, we add a clause containing the single literal r_m . We add a clause with the single literal $\neg r_m$ otherwise.

Results

We ran MiniMaxSAT and DPLL-S on a set of problems that varied based on the grid size N_g , the number of relations R and the number of objects O . All problems in the set were solvable and were created by randomly placing the requisite number of objects in a grid of the relevant size. Relations were then randomly chosen. Table 1 shows the number of variables and clauses for SAT and SAT-S problems for dif-

#relations	MiniMaxSAT	DPLL-S
5	0.9s	3.89s
10	2.49s	5.8s
15	3.49s	5.69s
20	4.19s	6.12
25	5.37s	6.19s
30	23.01s	1.46s
35	8.09s	6.52s
40	13.59s	4.76s

Table 3: MiniMaxSAT and DPLL-S runtimes for increasing number of relations

#objects	MiniMaxSAT	DPLL-S
3	0.43s	0.02s
4	1.46s	0.02s
5	1.33s	0.76s
6	1.95s	-

Table 4: MiniMaxSat and DPLL-S runtimes for increasing number of objects

ferent grid sizes. The size of a SAT-S problem depends on the number of relations that have to be satisfied. Problem sizes in SAT vary on the grid size, number of objects, number of relations and relation type. *Near* relations are more costlier as they result in many more clauses than the other four relation types. For this reason, we have broken down Tables 1 and 2 based on the number of *Near* relations as well. Table 2 shows the execution times for MiniMaxSAT and DPLL-S for increasing grid size. We stopped a run if it did not produce a result within 15 minutes. For comparison purposes, we used the propagate method to preprocess a SAT-S problem into a SAT problem whose runtimes are shown in the table as SAT-PP. The runtime for a single problem in SAT-PP is the sum of the times required to pre-process it into SAT and solve using MiniMaxSAT. From the table it is clear that a SAT-S problem requires virtually no time to run. This is despite the fact that the DPLL-S implementation is not geared towards efficiency. It was written using convenient but inefficient data structures and run in parallel with other processes on the system. The speedup obtained is purely due to the use of diagrammatic models.

Table 3 shows the comparison between MiniMaxSAT and DPLL-S for increasing number of relations given a standard grid size of 25x25 and 5 objects. For MiniMaxSAT, as the number of relations increase, the runtime increases. For DPLL-S, the increase is minimal because given a fixed number of objects, as the relations increase, the propagate method merely has to eliminate diagrams that do not satisfy relations instead of having to generate new diagrams.

Table 4 shows the comparison between MiniMaxSAT and DPLL-S for increasing number of objects with the grid size fixed at 25. This is where DPLL-S underperforms MiniMaxSAT. By the time the number of objects gets to six, there are so many diagrams being generated by the propagate method that it becomes impossible for the algorithm to

finish within the 15 minute limit. Since the number of diagrams generated increases exponentially, even an efficient implementation of the current DPLL-S algorithm would not be able to solve problems with more than 7 or 8 objects. There are, however, approaches that could allow us to effectively deal with this problem. Our current work is focused on maintaining a single diagram rather than the entire set of diagrams. As spatial constraints arise, this diagram is manipulated to satisfy these constraints. This strategy, while preserving completeness will provide better overall scalability at the expense of run times with small number of objects.

Conclusion

The satisfiability approach to problem solving has shown great promise in recent years. However, the effects of propositionalizing space, makes the satisfiability approach to spatial reasoning expensive. In this work, we have shown how the satisfiability approach can be augmented with the use of diagrammatic models to reduce the problem space. Our approach utilizes diagrams to represent space as discrete regions instead of individual points. This leads to savings in both problem size and solution times. We introduced a complete solver called DPLL-S, a variation of DPLL, and evaluated it against a current SAT solver MiniMaxSAT in a spatial reasoning problem. Our approach greatly reduced the number of variables and clauses in the formula and led to nearly instantaneous runtimes in many cases for solving the placement problem. One of the problems with the current DPLL-S approach is the explosions in the number of diagrams as a function of the number of objects and relations. In future work, we address these concerns and compare our algorithm against SMTs such as QF-IDL.

References

- Cotton, S. 2005. Satisfiability Checking With Difference Constraints. Master’s thesis, IMPRS Computer Science.
- DeMoura, L., and Rue, H. 2002. Lemmas on demand for satisfiability solvers. In *Proceedings of the Fifth International Symposium on the Theory and Applications of Satisfiability Testing (SAT)*, 244–251.
- Heras, F.; Larrosa, J.; and Oliveras, A. 2008. MiniMaxSAT: An efficient Weighted Max-SAT Solver. *Journal of Artificial Intelligence Research* 31:1–32.
- Randell, D. A.; Cui, Z.; and Cohn, A. G. 1992. A spatial logic based on regions and connection. In *Proceedings 3rd International Conference on Knowledge Representation and Reasoning*.
- Wintermute, S., and Laird, J. 2007. Predicate projection in a bimodal spatial reasoning system. In *Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence*. Vancouver, Canada: Morgan Kaufmann.

A Theoretical Framework to Formalize AGI-Hard Problems

Pedro Demasi*

Jayme L Szwarcfiter*[†]

Adriano J O Cruz[†]

Abstract

The main goal of the Artificial General Intelligence field (AGI) to create “human level intelligence” is known as a very ambitious one (Hut04). On the way to the field development there are many difficult problems to solve, like natural language translation, for example, which seem to share some “hardness” properties. The terms “AI-Complete” and “AI-Hard”, by analogy with the terms “NP-Complete” and “NP-Hard” from computational complexity theory (CLRS01), have been informally used to classify them although there are also works that propose some kind of formal definition (SA07), (vABHL03). This work proposes a theoretical framework with formal definitions to distinguish these problems and discuss its use in practical applications and how their properties can be used in order to achieve improvements in the AGI field.

Introduction

In order to achieve a human level intelligence, it is clear that many “human” problems must be solved on the way to it. Some tasks, like natural language understanding, are easy to be solved by humans. Even other tasks which may require some kind of prior training, like text translation between two natural languages, are also routinely solved by humans.

The lack of formal definition for such problems is still a huge barrier for computationally solving them. This work proposes a formal framework to classify which problems can be considered as human-level intelligence bounded and which can not.

We will thus distinguish the problems in these different classes:

- **Non AGI-Bound:** problems that are not of AGI interest. Although they may be “hard” in computational sense, they are not in the scope of AGI study.
- **AGI-Bound:** problems that require some kind of human-level intelligence to be properly solved.

*Universidade Federal do Rio de Janeiro, Cidade Universitaria, Programa de Engenharia de Sistemas e Computacao, Bloco H, Rio de Janeiro/RJ, Brazil, 21941-972

[†]Universidade Federal do Rio de Janeiro, Cidade Universitaria, Nucleo de Computacao Eletronica, Rio de Janeiro/RJ, Brazil, 20001-970

- **AGI-Hard:** problems that are at least as hard as any AGI-Bound problem.

Human Solvers and Human Oracles

The boolean set \mathbb{B} will be used as $\mathbb{B} = \{0, 1\}$ and $\mathbb{B} = \{\text{no}, \text{yes}\}$ interchangeably without any loss of generality.

Definition 1. A **solver** is a black box which can solve any problem a Turing Machine (TM) or a human can solve, using constant time. Formally, a solver will be a function $f_s : \mathbb{N} \rightarrow \mathbb{N}$ such that given an input $x \in \mathbb{N}$ it will give an output $y \in \mathbb{N}$ as the answer. The output y given by the solver is called an **acceptable** answer to the input x .

Definition 2. We say that $y \in \mathbb{N}$ is an **acceptable** output for input $x \in \mathbb{N}$ if, and only if, $\exists f_s$ such that $f_s(x) = y$.

Definition 3. We say that $y \in \mathbb{N}$ is an **incorrect** output for input $x \in \mathbb{N}$ if, and only if, $\nexists f_s$ such that $f_s(x) = y$.

Definition 4. An **oracle** to a problem \mathcal{P} is a function $f : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{B}$.

Definition 5. A **valid** oracle for a problem \mathcal{P} will always answer **yes** to an input pair (x, y) if y is a **correct** output for x . It will always answer **no** to an input pair (x, y) if y is an **incorrect** output for x . For every pair (x, y) such that y is an **acceptable** output for x , but not correct, the oracle may answer either **yes** or **no**.

Definition 6. The set $\Omega^{\mathcal{P}}$ is the set of all valid oracles for the problem \mathcal{P} .

Definition 7. We say that $x \in \mathbb{N}$ is a valid input for oracle $f : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{B}$ if, and only if, $\exists y \in \mathbb{N}$, $f(x, y) = 1$.

Definition 8. For a given problem \mathcal{P} , $y \in \mathbb{N}$ is a **correct** output for input $x \in \mathbb{N}$ if, and only if, $\forall f \in \Omega^{\mathcal{P}}$, $f(x, y) = 1$. This definition implies that every correct output is also an acceptable one.

Properties and Operation of Valid Oracles and their Sets

Definition 9. Two oracles are equal, $f_a = f_b$ if, and only if, $\forall x, y \in \mathbb{N}$, $f_a(x, y) = f_b(x, y)$

Definition 10. The intersection of two oracles f_a and f_b , represented as $f_c = f_a \cap f_b$, is $f_c = f_a(x, y) \wedge f_b(x, y) \forall x, y \in \mathbb{N}$

Definition 11. The union of two oracles f_a and f_b , represented as $f_c = f_a \cup f_b$, is $f_c = f_a(x, y) \vee f_b(x, y) \forall x, y \in \mathbb{N}$

Property 1. The set $\Omega^{\mathcal{P}}$ of valid oracles over problem \mathcal{P} is closed under intersections and unions.

Definition 12. A **master oracle** f_{Ω} for a problem \mathcal{P} is the union of all elements of $\Omega^{\mathcal{P}}$. Thus $f_{\Omega} = \bigcup_{f \in \Omega^{\mathcal{P}}} f$.

Definition 13. A **trivial oracle** f_{\emptyset} for a problem \mathcal{P} is the intersection of all elements of $\Omega^{\mathcal{P}}$. Thus $f_{\emptyset} = \bigcap_{f \in \Omega^{\mathcal{P}}} f$.

Definition 14. Let $\overline{f_a}$ be the complement of f_a , it is such that $f_a \cup \overline{f_a} = f_{\Omega}$ and $f_a \cap \overline{f_a} = f_{\emptyset}$.

Property 2. The set $\Omega^{\mathcal{P}}$ of valid oracles over problem \mathcal{P} is closed under complement.

Definition 15. We say that $f_a \geq f_b$ (meaning that f_a dominates f_b) if, and only if, for every valid input $x \in \mathbb{N}$ of f_b , $\forall y \in \mathbb{N}$, $f_a(x, y) = f_b(x, y)$

Property 3. Mutual dominance between oracles is such that $f_a \geq f_b, f_b \geq f_a \leftrightarrow f_a = f_b$

AGI-Boundness and AGI-Hardness

Definition 16. The cardinality of $\Omega^{\mathcal{P}}$ is such that $|\Omega^{\mathcal{P}}| = \prod_{x=1}^{|\mathbb{N}|} 2^{|\mathcal{A}_x^{\mathcal{P}}|} = 2^{\sum_{x=1}^{|\mathbb{N}|} |\mathcal{A}_x^{\mathcal{P}}|}$

Definition 17. A problem \mathcal{P} is Non AGI-Bound if, and only if, $|\Omega^{\mathcal{P}}| = 1$

Definition 18. Let the set $\mathcal{A}_x^{\mathcal{P}}$ be such that $\forall y \in \mathcal{A}_x^{\mathcal{P}}, f_{\Omega}(x, y) \otimes f_{\emptyset}(x, y) = 1$ (\otimes means logical exclusive-or).

Definition 19. A problem \mathcal{P} is AGI-Bound if, and only if, $|\Omega^{\mathcal{P}}| > 1$

Theorem 1. If the summation of cardinalities of $\mathcal{A}_x^{\mathcal{P}}$ is equal to the cardinality of \mathbb{N} , then the cardinality of $\Omega^{\mathcal{P}}$ is equal to \mathbb{R} , that is $\sum_{x=1}^{|\mathbb{N}|} |\mathcal{A}_x^{\mathcal{P}}| = |\mathbb{N}| \rightarrow |\Omega^{\mathcal{P}}| = |\mathbb{R}|$

Corollary 1. If there is at least one $x \in \mathbb{N}$ such that the cardinality of $\mathcal{A}_x^{\mathcal{P}}$ is equal to the cardinality of \mathbb{N} , then the cardinality of $\Omega^{\mathcal{P}}$ is equal to \mathbb{R} , that is $\exists x \in \mathbb{N}, |\mathcal{A}_x^{\mathcal{P}}| = |\mathbb{N}| \rightarrow |\Omega^{\mathcal{P}}| = |\mathbb{R}|$

Corollary 2. Let the set $\mathcal{B} = \{x, |\mathcal{A}_x^{\mathcal{P}}| > 0\}$, $|\mathcal{B}| = |\mathbb{N}| \rightarrow |\Omega^{\mathcal{P}}| = |\mathbb{R}|$

Using Definition 16, we have by Theorem 1 that $|\Omega^{\mathcal{P}}| = 2^{|\mathbb{N}|} = |\mathbb{R}|$. A simple way to see that $2^{|\mathbb{N}|} = |\mathbb{R}|$ is to realize that this is equivalent to have a countable infinite number of binary digits. Thus we can write any number of \mathbb{R} in binary base, and therefore we have an one-to-one correspondence.

Definition 20. A problem \mathcal{P} is AGI-Hard if, and only if, $|\Omega^{\mathcal{P}}| = |\mathbb{R}|$

Definition 21. Given an oracle $f : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{B}$ and a problem \mathcal{P} , return yes if $f \in \Omega^{\mathcal{P}}$ and no otherwise.

Definition 22. Given problems \mathcal{P} and \mathcal{Q} we define as *weak dominance* $\mathcal{P} \succeq \mathcal{Q} \leftrightarrow \exists f \in \mathcal{P}, \forall f' \in \mathcal{Q}, f \geq f'$

Definition 23. Given problems \mathcal{P} and \mathcal{Q} we define as *strong dominance* $\mathcal{P} \geq \mathcal{Q} \leftrightarrow \forall f \in \mathcal{P}, \forall f' \in \mathcal{Q}, f \geq f'$

Theorem 2. A condition for a problem that is not dominated $\exists x \in \mathbb{N}, |\mathcal{A}_x^{\mathcal{P}}| > 1 \rightarrow \nexists \mathcal{Q}, \mathcal{Q} \geq \mathcal{P}$

Theorem 2 says that if a problem \mathcal{P} has at least one input with more than one strictly acceptable output then there is no problem \mathcal{Q} such that \mathcal{Q} strongly dominates \mathcal{P} .

When we are dealing with traditional computational complexity we are usually concerned with the amount of basic steps our algorithm will perform in function of the size of the input. For AGI-Bound, however, the complexity can be measured in terms of the problem ambiguity, which is the size of the valid oracle set, $|\Omega|$.

Of course even if we can deal with ambiguity reduction there will be still the need of some time and space efficiently. It would be worthless if we are able to solve an AGI-Hard problem but not in an acceptable time frame or if we have no storage capacity for it. These concerns become more important when implementing any methods of AGI-Hard problems and there is the need for further development in this area also.

Further Work and Conclusion

This work presented a new theoretical framework to formally define what AGI problems are, which are “hard” problems and briefly outlined a way of interpreting AGI problems complexity.

Some points that will be addressed in future works will be a definition of “AI” problems on the set of Non AGI-Bound problems as dominated problems by AGI-Hard ones, the link between those kind of problems and a better definition of their relations.

AGI problems complexity development should be a great help to better solving some AGI-Hard problems, and the results of experimental application will show that.

The use of human computation in light of the theoretical model presented in this paper should also be investigated as it still can lead to very encouraging results.

In summary, we believe that the theoretical framework introduced in this paper, and to be further and deeper developed in future works, contributes to the AGI field research, helping to improve practical results and stimulating novel theoretical discussions.

References

- [CLRS01] T. Cormen, C. Leiserson, R. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, second edition, 2001.
- [Hut04] M. Hutter. *Universal Artificial Intelligence: Sequential Decisions based on Algorithmic Probability*. Springer, Berlin, 2004.
- [SA07] D. Shahaf and E. Amir. Towards a theory of ai-completeness. In *8th International Symposium on Logical Formalizations of Commonsense Reasoning*, 2007.
- [vABHL03] L. von Ahn, M. Blum, N. Hopper, and J. Langford. Captcha: Using hard ai problems for security. In *In Proceedings of Eurocrypt*, pages 294–311. Springer-Verlag, 2003.

Cognitive Architecture Requirements for Achieving AGI

John E. Laird*, Robert E. Wray III**

*Division of Computer Science and Engineering, University of Michigan, Ann Arbor, MI 48109-2121

**Soar Technology, Inc., 3600 Green Court, Suite 600, Ann Arbor, MI 48105

Abstract

We outline eight characteristics of the environments, tasks, and agents important for human-level intelligence. Treating these characteristics as influences on desired agent behavior, we then derive twelve requirements for general cognitive architectures. Cognitive-architecture designs that meet the requirements should support human-level behavior across a wide range of tasks, embedded in environment similar to the real world. Although requirements introduced here are hypothesized as necessary ones for human-level intelligence, our assumption is the list is not yet sufficient to guarantee the achievement of human-level intelligence when met. However, attempts to be explicit about influences and specific requirements may be more productive than direct comparison of architectural designs and features for communication and interaction about cognitive architectures.

Introduction

This paper explores requirements on cognitive architectures for artificial general intelligence. The goal of the analysis is to determine the requirements for cognitive architectures that support the full-range of human-level intelligent behavior. Although many different architectures have been proposed (and some built), understanding the relative strengths and weaknesses of these architectures and their unique contributions to the pursuit of human-level intelligence has proven elusive, whether via analytic comparison (e.g., Anderson & Lebiere, 2003; Jones and Wray, 2006) or empirical comparisons on task performance (e.g., Gluck & Pew. 2005).

However, as suggested by Cohen (1995), three influences determine an agent's behavior: the agent's structure, its environment, and tasks.¹ Given the diversity of environments and tasks, we are not attempting to create architectures that are necessarily the best or even sufficient for *all* possible environments and *all* possible tasks. We assume that agents exist in an environment and pursue tasks similar to those we find in the world we inhabit. The challenge is to take advantage of the structure of the environment and tasks in our architecture design, while avoiding optimizations that apply to only a subset of tasks. For specific problems, specialized architectures can be more appropriate (e.g., Deep Blue for chess, Campbell, Hoane & Hsu 2002).

Figure 1 illustrates how the characteristics of the environment, tasks, and agent structure determine a set of requirements for a cognitive architecture. These requirements in turn are the basis for a specific architecture design. Cognitive architectures must provide a comprehensive computational story that puts all the pieces of intelligence together from end to end.

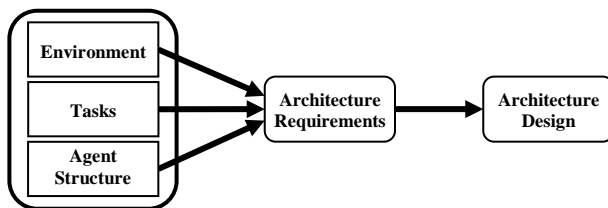


Figure 1: Influences on architecture design.

In practice, researchers have typically focused on communicating the architectural design of their systems and its performance on specific tasks rather than motivating the design via specific requirements. We propose to orient future discussion around requirements rather than specific designs. There will be two immediate benefits to this approach. First, it makes little sense to compare architectures (as works in progress) when they share few requirements. If one architecture attempts to satisfy a requirement that all decisions must be made in bounded time, whereas another is developed independent of that requirement, we would expect to see very different approaches that would be difficult, if not meaningless to compare. Being explicit about requirements will make it easier to see what “spaces” architectures are attempting to occupy – what environments and problems they are appropriate for. Secondly, because human-level intelligence is so broad, there is no existing list of necessary and sufficient requirements of AGI. This paper, drawing from our experience, proposes an initial list of these requirements. We expect it to be refined, extended, and corrected via interaction with other researchers.

We recognize this attempt is not novel. John Anderson took a step in this direction with the design of specific components of ACT-R using a rational analysis (Anderson, 1990). He determined optimal methods for primitive architectural functions, such as retrieving an item from declarative memory given the expected use of that memory in the future. This revolutionized his design process and led to significant advances in ACT, including the development of a new process for retrieving items from long-term declarative memory. Although rational analysis

¹ We use “task” for any type of problem, goal, drive, or reward that provides direction for agent behavior.

is useful for designing the performance of specific components, it is difficult to apply to the specification of a complete cognitive architecture as it does not specify what components there should be, or how they combine together to provide general intelligent behavior.

The analysis also builds on previous descriptions of evaluation criteria for cognitive architectures (Langley, Laird & Rogers, 2009; Laird et al. 2009; Laird et al., 1996; Laird, 1991) and theories of cognition (Anderson & Lebiere, 2003; Newell, 1990). For example, previously identified criteria include a mixture of constraints on behavior (flexible behavior, real-time performance) architecture (support vast knowledge bases), and underlying technology (brain realization) (Newell, 1990, Anderson & Lebiere, 2003). We separate characteristics of the environment, tasks, agent structure, and the behavior of an agent, which are described below, and then use them to derive requirements for cognitive architectures.

Environment, Task, and Agent Characteristics

In this section, we list characteristics of environments, tasks, and agents that lead to requirements for architectures that support human-level intelligent agents. Some of these characteristics are obvious, or so ingrained in the literature that they are rarely made explicit, such as the existence of regularities at different time scales in the environment. Some of these are characteristics of one of the three components, independent of the others, but many of them are characteristics of interactions between two or even all three. The interactions are important because the characteristics of an environment are only important to the extent they influence the agent's ability to pursue its tasks.

C1. ENVIRONMENT IS COMPLEX WITH DIVERSE INTERACTING OBJECTS

The world is large and complex. Agents can usefully interpret the environment as if it consists of independent objects (together with materials that do not have object-like structure, such as air, water, and sand). There are many objects and the objects interact with each other (i.e., via physics). Objects have numerous diverse properties.

C2. ENVIRONMENT IS DYNAMIC

The agent's environment can change independently of the agent so that the agent does not determine the state of the environment and the agent must respond to the dynamics of the world. Because the world can change while an agent is reasoning, an agent must be able to respond quickly relative to the dynamics of the environment. Moreover, the dynamics of the environment are so complex that an agent cannot always accurately predict future states in detail.

C3. TASK-RELEVANT REGULARITIES EXIST AT MULTIPLE TIME SCALES

An environment, while it may be complex and dynamic, it is not arbitrary. The environment is governed by laws of interaction that are constant, often predictable, and lead to

recurrence and regularity that impact the agent's ability to achieve goals. Regularities exist at a variety of time scales.

C4. OTHER AGENTS IMPACT TASK PERFORMANCE

The agent is not alone, and must interact with other agents in pursuit of its goals. Other agents may help or hinder the agent's achievement of its tasks. The agent can communicate with the other agents to share knowledge, indicate intent, etc. In addition, some agents have similar structure and capabilities to the agent (similar perception, action, and mental capabilities), making it possible to learn from other agents by observing the methods they use for solving problems. This characteristic is a special case of C1, C2, and C3, but has sufficient impact on the structure of agents to warrant distinct enumeration.

C5. TASKS CAN BE COMPLEX, DIVERSE, AND NOVEL

A general, intelligent agent must be able to work on a diverse set of novel, complex tasks. Tasks can interact so that in some cases, achieving one task aids in achieving another, while in other cases, achieving one makes it more difficult to achieve another. Tasks can also vary in the time scales required to achieve them, where the agent must achieve some tasks at close to the timescale of relevant changes in the environment, while others tasks can require extended behavior over time.

C6. AGENT/ENVIRONMENT/TASK INTERACTIONS ARE COMPLEX AND LIMITED

There may be many regularities in the environment, but they are only relevant if they can be detected and influence the agent's ability to perform its tasks. Thus, an agent has sufficient sensory capabilities that it can detect (possibly only through extensive learning) task-relevant regularities in the environment. An agent also has many mechanisms for acting in the environment in order to pursue a task. Although sensing and action modalities can be extensive, they are limited. The environment is partially observable, both from inherent physical limits in the sensors and the size of the environment. Sensors have noise and can be occluded by objects, have limited range, etc. making the agent's perception of its environment incomplete and uncertain. The agent's actions must obey the physical limitations of the environment. For example, actions usually take time to execute and have limited extent.

C7. AGENT COMPUTATIONAL RESOURCES ARE LIMITED

The agent has physical limits on its computational resources relative to the dynamics of the environment. The agent is unable to perform arbitrary computation in the time it has available to respond to the environment. Thus, an agent has *bounded rationality* (Simon, 1969) and cannot achieve perfect rationality (or universal intelligence, Legg & Hutter, 2007) in sufficiently complex environments and tasks when it has large bodies of knowledge.

C8. AGENT EXISTENCE IS LONG-TERM AND CONTINUAL

The agent is always present in its environment and it needs to actively pursue core tasks (such as self-protection) related to its survival. The agent may act to position itself

so that the dynamics of the environment have little impact on it for extended times (e.g., hide in a protected area), but it has no guarantee that those efforts will be successful. Further, the agent has a long-term existence relative to its primitive interactions with its environment. Its activity extends indefinitely across multiple tasks, and possibly multiple instances of the same task.

Architectural Requirements

Based on the characteristics of environments, tasks, and agents presented in the previous section, we derive the following requirements for cognitive architectures related to knowledge acquisition, representation, and use. Our goal is to generate a list that is necessary, such that all human-level agents must meet these requirements and that it is sufficient, such that meeting these requirements guarantees human-level behavior. The requirements we derive do not include criteria related to how well they model human behavior, nor the ease with which humans can create, debug, maintain, or extend agents developed in cognitive architectures. We also have not included criteria related to properties of the theory underlying the architecture, such as parsimony (Cassimatis, Bello, & Langley, 2008).

R0. FIXED STRUCTURE FOR ALL TASKS

An individual agent adapts to its environment not through changes in its architecture but through changes in knowledge. Architectures should not depend on parameters that are tuned to improve performance on a new task; although parameters can be useful for introducing variation across agents. Architectures also should not allow escape to a programming language for task-specific extensions. The rationale for this requirement is that environmental regularities exist [C3] at time scales that approach or exceed the life of the agent [C8] that are worth capturing in a fixed architecture.

R1. REALIZE A SYMBOL SYSTEM

The consensus in AI and cognitive science is that in order to achieve human-level behavior, a system must support universal computation. Newell (1990) makes the case that symbol systems provide both sufficient and necessary means for achieving universal computation; that is, a symbol system is capable of producing a response for every computable function. Possibly most important, symbol systems provide flexibility. In particular, they provide the ability to manipulate a description of some object in the world “in the head” without having to manipulate the object in the real world. Symbol structures also provide arbitrary composability to match the combinatoric complexity and regularity of the environment [C1, C3]. Thus, structures encountered independently can be combined later to create novel structures never experienced together [C5]. This generative capability is what we do when we combine letters or sounds to make new words, and when we combine words to make new sentences, and so on. Symbol systems also allow us to accept instructions from another agent and then use those

instructions later to influence behavior (interpretation) – providing additional flexibility and more generality – so that not everything must be programmed into a symbol system beforehand. In addition, symbols are required for communication that does not cause the meaning to be directly experienced by the agent [C4]. For example, striking someone directly causes an experience in another agent, while a verbal threat involves the transmission of symbols that require interpretation.

Requiring that the agent realize a symbol system does not imply that symbolic processing must be implemented directly via some symbolic knowledge representation. Neural and connectionist models can obviously support human-level behavior. Rather, this requirement posits that such approaches must implement symbol systems to some degree (Barsalou, 2005).

R2. REPRESENT AND EFFECTIVELY USE MODALITY-SPECIFIC KNOWLEDGE

Although pure symbol systems support universal computation, they rely on modality-independent methods for representing and reasoning to achieve universality and complete composability. However, complete composability is not always necessary. Modality-specific representations can support more efficient processing through regularities [C3] in sensory processing [C6]. For example, some representations and associated processes for visual input have qualitatively different computational properties for image operations. Examples include rotation and inversion, and detecting and reasoning about spatial relations. For some tasks [C5] given limited computational resources [C7], modality-specific representations are necessary for achieving maximal efficiency, especially in tasks that require real-time performance [C2].

R3. REPRESENT AND EFFECTIVELY USE LARGE BODIES OF DIVERSE KNOWLEDGE

The agent must be able to represent and use large bodies of knowledge. This wealth of knowledge that arises from the complexity of the environment [C1] and its associated regularities [C3], the variety of tasks the agent must pursue [C5], its complex interaction with the environment [C6], and the agent’s continual existence [C8]. This knowledge is diverse, including memories of experiences, facts and beliefs, skills, and knowledge about other agents [C4].

R4. REPRESENT AND EFFECTIVELY USE KNOWLEDGE WITH DIFFERENT LEVELS OF GENERALITY

The agent must represent and use general knowledge that takes advantage of the environmental regularities [C3]. The agent must also be sensitive to details of its current situation and its relationship to its tasks. These details are ubiquitous in complex [C1], dynamic [C2] environments where the agent can have many tasks [C5].

R5. REPRESENT AND EFFECTIVELY USE DIVERSE LEVELS OF KNOWLEDGE

An agent must be able to take advantage of whatever knowledge is available. For novel tasks and environments,

its knowledge is limited, and even for familiar tasks and environments, its knowledge may be incomplete, inconsistent, or incorrect. If there is extensive knowledge available for a task, the agent must be able to represent and effectively use it. There are regularities in the environment worth knowing [C3], the complexity of an agent's limited sensing of its environment [C6], the complexity of its environment and tasks [C5], and limits on its computational resources [C7]. Planning systems often fail on this requirement. They often have a required and fixed set of input knowledge (the task operators and a declarative description of the goal). Without this knowledge, they are unable to attempt the problem. Further, if additional knowledge is available (such as knowledge about the likelihood of an operator leading to the goal), the planner is often unable to use it to improve behavior.

R6. REPRESENT AND EFFECTIVELY USE BELIEFS INDEPENDENT OF CURRENT PERCEPTION

The agent must be able to represent and reason about situations and beliefs that differ from current perception. Perceptual information is insufficient because perception is limited [C6], the environment is dynamic [C2], and there are regularities in the environment worth remembering [C3] for task completion [C5]. Thus, the agent must be able to maintain history of prior situations as well as the ability to represent and reason about hypothetical situations, a necessary component of planning. An agent that satisfies this requirement can make decisions based not just on its current situation, but also on its memory of previous situations and its prediction of future situations.

R7. REPRESENT AND EFFECTIVELY USE RICH, HIERARCHICAL CONTROL KNOWLEDGE

The agent must have a rich representation for control, because the actions it can perform are complex [C6]. Because of the dynamics of the environment [C2], and the multiplicity of the tasks playing out at multiple time scales [C5], some actions may need to occur in rapid sequence while others may need to execute in parallel. To keep up with a rapidly changing environment [C2] with limited computational resources [C7], the agent must take advantage of the structure of regularities of the environment [C3], maximizing the generality of the knowledge it encodes because of the complexity and variability of the environment and the agent's tasks [C1, C5]. This often means organizing knowledge about actions hierarchically. The agent can then decompose some of its actions into sequences of simpler actions, using the context of higher-level actions to constrain choices and reduce the knowledge required to generate action.

R8. REPRESENT AND EFFECTIVELY USE META-COGNITIVE KNOWLEDGE

In addition to the different types of knowledge discussed above, it is sometime necessary for an agent represent and use knowledge about itself and about its own knowledge (meta-knowledge). An agent invariably faces novel tasks [C5] where its task knowledge and/or computational resources [C7] are insufficient to determine the appropriate

behavior given the environmental complexity [C1], but in which there are regularities it can take advantage of [C3]. In these situations, an intelligent agent can detect its lack of task knowledge, and then use meta-knowledge to acquire new task knowledge. An agent can use other types of meta-cognitive knowledge to set its own goals and to direct future behavior in preparation for tasks, events, and situations that it expects to arise in the future. This is in response to the characteristics listed above as well as to the fact that the agent exists beyond a single task or problem [C8]. The exact range of necessary meta-cognitive knowledge is unclear – some appears to be necessary, but complete meta-cognitive knowledge is not required, at least in humans. Humans do not always know exactly what they know and often only discover what they know when they are put in a situation where that knowledge is useful.

R9. SUPPORT A SPECTRUM OF BOUNDED AND UNBOUNDED DELIBERATION

At one extreme, the agent must be able to react with bounded computation [C5] for tasks with time constraints close to those of the dynamics of the environment [C2]. It cannot reason or plan from first principles for all tasks because of inherent limits to its computational resources [C7]. At its most primitive level, the absolute time to respond must be bounded by the environmental dynamics for some subclass of its responses. Reactivity would be sufficient if the agent's knowledge of the environment and other agents was complete and correct and encoded for bounded access below the level of dynamics of the environment. However, in general, that is not possible because of the complexity of the environment [C1], the diversity of tasks [C5] and the limitations on environmental interaction [C6]. Moreover, at the other extreme, when there are sufficient computational resources available relative to the dynamics of the environment and task, the agent should have the ability to compose novel responses based on its knowledge that takes advantage of regularities in the tasks and environment [C3]. This composition is the basis for planning and it takes time, but allows the agent to integrate its diverse and potentially large bodies of knowledge for novel situations [R1-R8]. In between these two extremes, the agent must balance the tradeoff between deliberation and reaction based on its knowledge of the situation.

R10. SUPPORT DIVERSE, COMPREHENSIVE LEARNING

An agent with long-term existence [C8] requires different learning mechanisms when exposed to diverse environments [C1] and tasks [C5] having complex interactions [C6]. Learning takes advantage of regularities [C3], some of which can be extracted from a single situation in which all of the information is available at the same time, whereas in others; the information may be spread across time. Although general learning mechanisms exist, they are invariably biased toward specific types of knowledge that are available to the agent in different ways and often at different time scales. Moreover, a general cognitive architecture should be able to learn all the types

of task-specific knowledge it represents and uses, a property we call the *learning completeness principle*. A significant component of our own research is to explore what types of knowledge different learning mechanisms can contribute to achieving learning completeness.

R11.SUPPORT INCREMENTAL, ONLINE LEARNING

An agent with long-term existence [C8] that is in a complex active environment [C1, C2] with regularities [C3] must learn and modify its knowledge base so that it can take advantage of the environmental regularities [C3] when they are available. Once the experience has happened, it is gone. Only the information that the agent itself stores while it is behaving is available to guide its future behavior. This is not to suggest that an agent cannot recall prior situations and perform additional analysis at some future time [R6]; however, some primitive learning mechanism must store away the experience for that future, more deliberative learning. Moreover, the mechanisms for storing and retrieving those experiences must scale as more and more experiences are captured. Incremental learning incorporates experiences when they are experienced.

One implication of this requirement (together with the need for large bodies of knowledge) is that new knowledge must be acquired at low, bounded computational cost in real time; learning should not disrupt the agent's ongoing behavior by significantly slowing overall processing and negatively impacting its ability to react to its environment.

Summary

In this paper, we outlined characteristics of the environments, tasks, and agents important for human-level intelligence and, from these characteristics, derived requirements for general cognitive architectures. Architectural designs following from meeting the requirements should support human-level behavior across a wide range of tasks, embedded in environment similar to the real world. Figure 2 summarizes the analysis described

above. The figure highlights the dense connectivity between characteristics and requirements – no single characteristic is solely responsible for any requirement and no characteristic influences only a single requirement.

Many characteristics are necessary to derive most of their associated requirements because eliminating a characteristic allows for extreme simplification. Simple environments [C1] require only simple agents. There is no need to have large bodies of knowledge, no need for rich representations of action, and limited need to learn. An agent that only pursues simple well-known tasks [C5], or has unlimited computation [C7] can be much simpler than one that supports agents and tasks, in an environments with these characteristics. At the extreme is the requirement for task-relevant regularities [C3], which has universal impact because only with environmental regularities are knowledge, reasoning, learning, and architecture useful.

Discussion & Conclusion

The requirements we derived (R0-R11) define a rough design envelope for underlying architectures. However, the role of knowledge in agent development complicates attempts to match the achievement of specific requirements with specific architectural components. Behavior in an agent is the result of the interaction between knowledge and architecture; some requirements may be achieved through general knowledge combined with multiple architectural components. For example, many cognitive architectures do not have explicit architectural support for planning. Not including such architectural support simplifies these architectures, but requires encoding of knowledge representation(s) and algorithms for planning using architectural primitives. Achieving a requirement directly with the architecture allows for a more efficient implementation. Achieving a requirement in knowledge usually leads to a simpler architecture while providing more flexibility and the possibility of improving the capability through learning. This tension is analogous to

	C1 Complex Environ.	C2 Dynamic Environ.	C3 Task Regularities	C4 Social Environ.	C5 Complex Tasks	C6 Limited Interaction	C7 Limited Computation	C8 Long-term existence
R0 Fixed structure			X					X
R1 Symbol system	X		X	X	X			
R2 Modularity knowledge		X	X		X	X	X	
R3 Large bodies knowledge	X		X	X	X	X		X
R4 Levels of generality	X	X	X		X			
R5 Levels of knowledge			X		X	X	X	
R6 Non-perceptual represent.		X	X		X	X		
R7 Rich action representations	X	X	X		X	X	X	
R8 Meta-cognitive knowledge	X		X		X		X	X
R9 Spectrum of deliberation	X	X	X		X	X	X	
R10 Comprehensive learning	X		X		X	X		X
R11 Incremental learning	X	X	X				X	X

Figure 2: Connections between environment, task, and agent characteristics (C1-C8) and requirements (R0-R11).

RISC vs. CISC trade-offs in traditional computer architecture.

Our own hypothesis is that significant bodies of knowledge in combination with the architecture are required for many of the cognitive capabilities needed to achieve human-level performance. Examples include natural language processing, logical thinking, qualitative reasoning, and multi-agent coordination. However, the requirements listed above do not address what knowledge is necessary to support such capabilities, or how that knowledge is acquired and encoded. Thus, even if we create architectures that satisfy all of the listed requirements, we will still fall short of creating human-level agents until we encode, or the systems learn on their own, the content required for higher-level knowledge-intensive capabilities.

Even when we restrict ourselves to considering the requirements within the context of cognitive architecture independent of knowledge, it is difficult to evaluate the sufficiency of these requirements by examination alone. Many of the requirements are qualitative and vague, making them difficult to apply to existing architectures. For example, how do we judge whether an architecture supports sufficient levels of generality in its knowledge representations, or sufficient representations of meta-cognitive knowledge, or sufficiently comprehensive learning mechanisms? Thus, an important goal for future research in human-level agents is to refine these requirements as we learn more about the capabilities that are necessary for human-level behavior.

The current list of requirements emphasizes necessity and may be missing some yet to be discovered requirements that are needed to guarantee human-level behavior. These requirements may arise from interactions among the existing characteristics (C1-C8) or they may arise because of the existence of yet additional characteristics of agents, tasks, and environments that are relevant to achieving human-level intelligence.

Our own hypothesis is that one of the best ways to refine and extend these sets of requirements and characteristics is to develop agents using cognitive architectures that test the sufficiency and necessity of all these and other possible characteristics and requirements on a variety of real-world tasks. One challenge is to find tasks and environments where all of these characteristics are active, and thus all of the requirements must be confronted. A second challenge is that the existence of an architecture that achieves a subset of these requirements, does not guarantee that such an architecture can be extended to achieve other requirements while maintaining satisfaction of the original set of requirements. Usually there are too many potential interactions between architectural components to guarantee such an incremental approach. It is for these reasons that our own research is inspired by studies of human psychology. We know that the human cognitive architecture is sufficient for generating the behavior we seek from our agents, and if we build systems that capture

the core functionality of components of the human architecture, it is more likely that we will avoid dead ends in cognitive architecture development. Independent of what approach is used to develop a cognitive architecture, we propose that exploring how different architectures address (or do not address) these requirements, both theoretically and empirically, is our best chance to advance our knowledge of how cognitive architecture can support human-level intelligence.

References

- Anderson, J. R. (1990). *The Adaptive Character of Thought*, Hillsdale, NJ: Erlbaum.
- Anderson, J. R. & Lebiere, C. L. (2003). The Newell Test for a Theory of Cognition. *Behavioral & Brain Science* 26, 587-637.
- Barsalou, L. W. (2008). Grounded Cognition. *Annual Review of Psychology*, 59, 617-645.
- Campbell, M., Hoane, A. J., & Hsu, F. (2002). Deep Blue. *Artificial Intelligence*, 134(1-2) 57-83.
- Cassimatis, N.L., Bello, P., & Langley, P. (2008). Ability, Parsimony and Breadth in Models of Higher-Order Cognition. *Cognitive Science*. 33(8), 1304-1322.
- Cohen, P. R., (1995). *Empirical Methods for Artificial Intelligence*, Cambridge, MA: MIT Press.
- Gluck, K., & Pew, R., eds. (2005). *Modeling Human Behavior with Integrated Cognitive Architectures: Comparison, Evaluation, and Validation*. Lawrence-Erlbaum Associates, Mahwah, NJ.
- Jones, R. M., & Wray, R. E. (2006). Comparative Analysis of Frameworks for Knowledge-Intensive Intelligent Agents. *AI Magazine* 27, 57-70.
- Laird, J. E. (1991). Preface for Special Section on Integrated Cognitive Architectures. *SIGART Bulletin*, 2(12), 123.
- Laird, J. E., Wray, R. E. III, Marinier, R. P. III, & Langley, P. (2009). Claims and Challenges in Evaluating Human-Level Intelligent Systems, *Proceedings of the Second Conference on Artificial General Intelligence*.
- Laird, J. E., Pearson, D. J., Jones, R. M., & Wray, R. E. (1996). Dynamic Knowledge Integration During Plan Execution. *Papers from the 1996 AAAI Fall Symposium on Plan Execution: Problems and Issues*, 92-98. AAAI Press, Cambridge, MA.
- Langley, P., Laird, J. E., & Rogers, S. (2009). Cognitive Architectures: Research Issues and Challenges. *Cognitive Systems Research* 10(2), 141-160.
- Legg, S., & Hutter, M. (2007). Universal Intelligence: A Definition of Machine Intelligence. *Minds and Machines*. 17(4).
- Newell, A. (1990). *Unified Theories of Cognition*, Harvard University Press, Cambridge, Massachusetts.
- Simon, H. A. (1969). *The Sciences of the Artificial (First Edition)*, MIT Press.

Playing General Structure Rewriting Games

Łukasz Kaiser

Mathematische Grundlagen der Informatik
RWTH Aachen

Łukasz Stafiniak

Institute of Computer Science
University of Wrocław

Abstract

Achieving goals in a complex environment in which many players interact is a general task demanded from an AI agent. When goals of the players are given explicitly, such setting can be described as a multi-player game with complete information. We introduce a general model of such games in which states are represented by relational structures (hypergraphs), possibly with real-valued labels, and actions by structure rewriting rules. For this model, we develop an algorithm which computes rational strategies for the players. Our algorithm can be parametrized by a probabilistic evaluation function and we devise a general procedure for learning such evaluations. First tests on a few classical examples substantiate the chosen game model and our algorithm.

Introduction

As frustrated users know, a computer sometimes simply does not *want* to work. At other times, a car does not *want* to start. These phrases show how natural it is to ascribe *desires* and *goals* to active objects in an effort to understand them. Not only is it natural: it is arguably very useful as well. We encounter many complex objects in the environment, including ourselves and other people, and it is impossible to understand their function in detail. Knowing their goals and intentions, even vaguely, already allows to predict their actions to a useful degree.

In this paper, we present a modeling system in which explicitly given goals of multiple players define the dynamics. To run such a system, an algorithm making rational decisions for the players is necessary. Our main contribution is exactly such an algorithm, which gives reasonable results by default and can take probabilistic evaluation functions as an additional parameter. We also devise a general learning mechanism to construct evaluation functions. This mechanism is non-deterministic: the choice of the next hypothesis is delegated to an external function. Still, even instantiated with a very simple policy it manages to learn useful evaluations, at least in a few basic examples. Some components used in the presented algorithms may be of independent interest. One of them is a solver for an expressive logic, another one is a generalization of the Monte-Carlo playing algorithm with Upper Confidence bounds for Trees (UCT).

The UCT algorithm has already been used in a general game playing (GGP) competition. Cadia player [2], a pro-

gram using UCT, won the competition in 2007 demonstrating good performance of the UCT algorithm. Sadly, both the way of representing games in the GGP framework and the examples used in the competition lack true generality: These are either board games (e.g. connect-4, chess, go) or maze games (e.g. pac-man) described in a very basic prolog-like language [4]. There is neither a way to represent continuous real-time dynamics in the GGP framework, nor a way to define probabilistic choice. Moreover, in almost all examples it is possible to distinguish a fixed board and pieces moved by the players. Thus, for programs entering the GGP competition certain narrow board-game heuristics are crucial, which reduces their applications to AGI. We give both a general game model, representing states in a way similar to generalized hypergraphs [5] already used for AGI in OpenCog, and a general algorithm which is capable to learn useful patterns in specific situations without any fixed prior heuristic.

Organization. In the first two sections, we show how to represent the state of the world in which the agents play and their actions. The first section discusses discrete dynamics and the second one specifies how continuous values evolve. In the third section we introduce the logic used to describe patterns in the states. We finalize the definition of our model of games in the fourth section. Next, we proceed to the algorithmic part: we first describe the generalized UCT algorithm for playing games and then the learning procedure for evaluation functions. Finally, we present a few experimental results and conclude with perspectives on the applications of our modeling system and algorithms in AGI projects.

Discrete Structure Rewriting

To represent a state of our model in a fixed moment of time we use finite relational structures, i.e. labelled directed hypergraphs. A relational structure $\mathfrak{A} = (A, R_1, \dots, R_k)$ is composed of a universe A and a number of relations R_1, \dots, R_k . We denote the arity of R_i by r_i , so $R_i \subseteq A^{r_i}$. The *signature* of \mathfrak{A} is the set of symbols $\{R_1, \dots, R_k\}$.

The dynamics of the model, i.e. the way the structure can change, is described by *structure rewriting rules*, a generalized form of term and graph rewriting. Extended graph rewriting is recently viewed as the programming model of choice for complex multi-agent systems, especially ones with real-valued components [1]. Moreover, this form of rewriting is well suited for visual programming and helps to

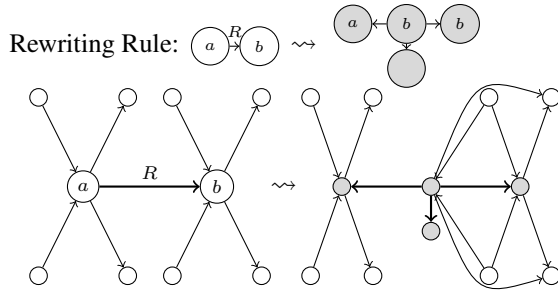


Figure 1: Rewriting rule and its application to a structure.

make the systems understandable.

In the most basic setting, a rule $\mathcal{L} \rightarrow_s \mathcal{R}$ consists of two finite relational structures \mathcal{L} and \mathcal{R} over the same signature and a partial function $s : \mathcal{R} \rightarrow \mathcal{L}$ specifying which elements of \mathcal{L} will be substituted by which elements of \mathcal{R} .

Let \mathcal{A}, \mathcal{B} be two structures, τ_e a set of relations symbols to be matched exactly and τ_h a set of relations to be matched only positively.¹ A function $f : \mathcal{A} \hookrightarrow \mathcal{B}$ is a (τ_e, τ_h) -embedding if f is injective, for each $R_i \in \tau_e$ it holds that $(a_1, \dots, a_{r_i}) \in R_i^{\mathcal{A}} \Leftrightarrow (f(a_1), \dots, f(a_{r_i})) \in R_i^{\mathcal{B}}$, and for $R_j \in \tau_h$ it holds that $(a_1, \dots, a_{r_j}) \in R_j^{\mathcal{A}} \Rightarrow (f(a_1), \dots, f(a_{r_j})) \in R_j^{\mathcal{B}}$. A (τ_e, τ_h) -match of the rule $\mathcal{L} \rightarrow_s \mathcal{R}$ in another structure \mathcal{A} is an (τ_e, τ_h) -embedding $\sigma : \mathcal{L} \hookrightarrow \mathcal{A}$. We define the result of an application of $\mathcal{L} \rightarrow_s \mathcal{R}$ to \mathcal{A} on the match σ as $\mathcal{B} = \mathcal{A}[\mathcal{L} \rightarrow_s \mathcal{R}/\sigma]$, such that the universe of \mathcal{B} is given by $(A \setminus \sigma(L)) \cup R$, and the relations as follows. A tuple (b_1, \dots, b_{r_i}) is in the new relation $R_i^{\mathcal{B}}$ if and only if either it is in the relation in \mathcal{R} already, $(b_1, \dots, b_{r_i}) \in R_i^{\mathcal{R}}$, or there exists a tuple in the previous structure, $(a_1, \dots, a_{r_i}) \in R_i^{\mathcal{A}}$, such that for each i either $a_i = b_i$ or $a_i = \sigma(s(b_i))$, i.e. either the element was there before or it was matched and b_i is the replacement as specified by the rule. Moreover, if $R_i \in \tau_e$ then we require in the second case that at least one b_i was already in the original structure, i.e. $b_i = a_i$. To understand this definition it is best to consider an example, and one is given in Figure 1.

Continuous Evolution

To model continuous dynamics in our system, we supplement relational structures with a number of labeling functions f_1, \dots, f_l , each $f_i : A \rightarrow \mathbb{R}$ (A is the universe).² Accordingly, each rewriting rule is extended by a system of ordinary differential equations (ODEs) and a set of right-hand update equations. We use a standard form of ODEs: $f_{i,l}^k = t(f_{i,l}^0, \dots, f_{i,l}^{k-1})$, where f_i are the above-mentioned functions, l can be any element of the left-hand side structure and f^k denotes the k -th derivative of f . The term $t(\bar{x})$ is constructed using standard arithmetic functions $+$, $-$, \cdot , $/$, natural roots $\sqrt[n]{\cdot}$ for $n > 1$ and rational numbers $r \in \mathbb{Q}$ in addition to the variables \bar{x} and a set of parameters \bar{p} fixed

¹In practice, we also allow some tuples in \mathcal{L} to be optional; this is a shorthand for multiple rules with the same right-hand side.

²In fact $f_i(a)$ is not in \mathbb{R} ; it is a function $\varepsilon \rightarrow (x, x + \delta)$, $\delta < \varepsilon$.

for each rule. The set of right-hand side update equations contains one equation of the form $f_{i,r} = t(\overline{f_{i,l}})$ for each function f_i and each r from the right-hand side structure.

Let $\mathcal{R} = \{(\mathcal{L}_i \rightarrow_{s_i} \mathcal{R}_i, \mathcal{D}_i, \mathcal{T}_i) \mid i < n\}$ be a set of rules extended with ODEs \mathcal{D}_i and update equations \mathcal{T}_i as described above. Given, for each rule in \mathcal{R} , a match σ_i of the rule in a structure \mathcal{A} , the required parameters \bar{p} and a time bound t_i , we define the result of a *simultaneous application* of \mathcal{R} to \mathcal{A} , denoted $\mathcal{A}[\mathcal{R}/\{\sigma_i, t_i\}]$, as follows.³

First, the structure \mathcal{A} evolves in a continuous way as given by the *sum* of all equations \mathcal{D}_i . More precisely, let \mathcal{D} be a system of differential equations where for each $a \in \mathcal{A}$ there exists an equation defining $f_{i,a}^k$ if and only if there exists an equation in some \mathcal{D}_j for $f_{i,l}^k$ for some l with $\sigma_j(l) = a$. In such case, the term for $f_{i,a}^k$ is the sum of all terms for such l , with each $f_{i,l}^m$ replaced by the appropriate $f_{i,\sigma_j(l)}^m$. Assuming that all functions f_i and all their derivatives are given at the beginning, there is a unique solution for these variables which satisfies \mathcal{D} and has all other, undefined derivatives set by the starting condition from \mathcal{A} . This solution defines the value of $f_{i,a}(t)$ for each $a \in \mathcal{A}$ at any time moment t .

Let $t^0 = \min_{i < n} t_i$ be the lowest chosen time bound and let i_0, \dots, i_k be all rules with this bound, i.e. each $t_{i_m} = t^0$. We apply each of these rules independently¹ to the structure \mathcal{A} at time t^0 . Formally, the relational part of $\mathcal{A}[\mathcal{R}/\{\sigma_i, t_i\}]$ is equal to $\mathcal{A}[\mathcal{L}_{i_0} \rightarrow_{s_{i_0}} \mathcal{R}_{i_0}/\sigma_{i_0}] \cdots [\mathcal{L}_{i_k} \rightarrow_{s_{i_k}} \mathcal{R}_{i_k}/\sigma_{i_k}]$ and the function values $f_i(a)$ are defined as follows. If the element a was not changed, $a \in \mathcal{A}$, then we keep the function value from the solution of \mathcal{D} , i.e. $f_i(a) = f_{i,a}(t^0)$. In the other case a was on the right-hand side of some rule, $a \in \mathcal{R}_m$. Let $f_{i,a} = t(\overline{f_{j,l}})$ be the equation in \mathcal{T}_m defining $f_{i,a}$. The new value of $f_i(a)$ is then computed by inserting the appropriate values for $f_{j,l}$ from the solution of \mathcal{D} into $t(\overline{f_{j,l}})$, i.e. $f_i(a) = t(\overline{y_{j,l}})$ where each $y_{j,l} = f_{j,\sigma_m(l)}(t^0)$.

Example. Let us define a simple two-dimensional model of a cat chasing a mouse. The structure we use, $\mathcal{A} = (\{c, m\}, C, M, x, y)$, has two elements c and m , unary relations $C = \{c\}$ and $M = \{m\}$ used to identify which element is which and two real-valued functions x and y . Both rewriting rules have only one element, both on the left-hand side and on the right-hand side, and the element is in C for the cat rule and in M for the mouse rule. The ODEs for both rules are of the form $x' = p_x, y' = p_y$, where p_x, p_y are parameters. The update equations just keep the left-hand side values, $x_r = x_l, y_r = y_l$. In this setting, simultaneous application of the cat rule with parameters p_x^c, p_y^c for time t^c and the mouse rule with parameters p_x^m, p_y^m for time t^m will have the following effect: The cat will move with speed p_x^c along the x -axis and p_y^c along the y -axis and the mouse analogously with p_x^m and p_y^m , both for time $t^0 = \min(t^c, t^m)$.

Logic and Constraints

The logic we use for specifying properties of states is an extension of monadic second-order logic with real-valued terms and counting operators. The main motivation for the

³Assume no two intersecting rules have identical time bounds.

choice of such logic is *compositionality*: To evaluate a formula on a large structure \mathcal{A} which is composed in a regular way from substructures \mathcal{B} and \mathcal{C} it is enough to evaluate certain formulas on \mathcal{B} and \mathcal{C} independently. Monadic second-order logic is one of the most expressive logics with this property and allows to define various useful patterns such as stars, connected components or acyclic subgraphs.⁴

In the syntax of our logic, we use first-order variables (x_1, x_2, \dots) ranging over elements of the structure, second-order variables (X_1, X_2, \dots) ranging over *sets* of elements, and real-valued variables ($\alpha_1, \alpha_2, \dots$) ranging over \mathbb{R} , and we distinguish boolean formulas φ and real-valued terms ρ :

$$\begin{aligned} \varphi &:= R_i(x_1, \dots, x_{r_i}) \mid x_i = x_j \mid x_i \in X_j \mid \rho <_\varepsilon \rho \mid \varphi \wedge \varphi \mid \\ &\quad \varphi \vee \varphi \mid \neg \varphi \mid \exists x_i \varphi \mid \forall x_i \varphi \mid \exists X_i \varphi \mid \forall X_i \varphi \mid \exists \alpha_i \varphi \mid \forall \alpha_i \varphi, \\ \rho &:= \alpha_i \mid f_i(x_j) \mid \rho \dot{+} \rho \mid \chi[\varphi] \mid \min_{\alpha_i} \varphi \mid \sum_{\bar{x} \mid \varphi} \rho \mid \prod_{\bar{x} \mid \varphi} \rho. \end{aligned}$$

Semantics of most of the above operators is defined in the well known way, e.g. $\rho + \rho$ is the sum and $\rho \cdot \rho$ the product of real-valued terms, and $\exists X \varphi(X)$ means that there exists a set of elements S such that $\varphi(S)$ holds. Among less known operators, the term $\chi[\varphi]$ denotes the characteristic function of φ , i.e. the real-valued term which is 1 for all assignments for which φ holds and 0 for all other. To evaluate $\min_{\alpha_i} \varphi$ we take the minimal α_i for which φ holds (we allow $\pm\infty$ as values of terms as well). The terms $\sum_{\bar{x} \mid \varphi} \rho$ and $\prod_{\bar{x} \mid \varphi} \rho$ denote the sum and product of the values of $\rho(\bar{x})$ for all assignments of elements of the structure to \bar{x} for which $\varphi(\bar{x})$ holds. Note that both these terms can have free variables, e.g. the set of free variables of $\sum_{\bar{x} \mid \varphi} \rho$ is the union of free variables of φ and free variables of ρ , minus the set $\{\bar{x}\}$. Observe also the ε in $<_\varepsilon$: the values $f(a)$ are given with arbitrary small but non-zero error (cf. footnote 2) and $\rho_1 <_\varepsilon \rho_2$ holds only if the upper bound of ρ_1 lies below the lower bound of ρ_2 .

The logic defined above is used in structure rewriting rules in two ways. First, it is possible to define a new relation $R(\bar{x})$ using a formula $\varphi(\bar{x})$ with free variables contained in \bar{x} . Defined relations can be used on left-hand sides of structure rewriting rules, but are not allowed on right-hand sides. The second way is to add *constraints* to a rule. A rule $\mathcal{L} \rightarrow_s \mathcal{R}$ can be constrained using three sentences (i.e. formulas without free variables): φ_{pre} , φ_{inv} and φ_{post} . In both φ_{pre} and φ_{inv} we allow additional constants l for each $l \in \mathcal{L}$ and in φ_{post} special constants for each $r \in \mathcal{R}$ can be used. A rule $\mathcal{L} \rightarrow_s \mathcal{R}$ with such constraints can be applied on a match σ in \mathcal{A} only if the following holds: At the beginning, the formula φ_{pre} must hold in \mathcal{A} with the constants l interpreted as $\sigma(l)$. Later, during the whole continuous evolution, the formula φ_{inv} must hold in the structure \mathcal{A} with continuous values changed as prescribed by the solution of the system \mathcal{D} (defined above). Finally, the formula φ_{post} must hold in the resulting structure after rewriting. During simultaneous execution of a few rules with constraints and with given time bounds t_i , one of the invariants φ_{inv} may cease to hold. In such case, the rule is applied at that moment of time, even before $t^0 = \min t_i$ is reached — but of course only if φ_{post} holds afterwards. If φ_{post} does not hold, the rule is ignored and time goes on for the remaining rules.

⁴We provide additional syntax (shorthands) for useful patterns.

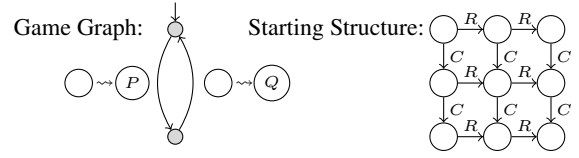


Figure 2: Tic-tac-toe as a structure rewriting game.

Structure Rewriting Games

As you could judge from the cat and mouse example, one can describe a structure rewriting game simply by providing a set of allowed rules for each player. Still, in many cases it is necessary to have more control over the flow of the game and to model probabilistic events. For this reason, we use labelled directed graphs with probabilities in the definition of the games. The labels for each player are of the form:

$$\lambda = (\mathcal{L} \rightarrow_s \mathcal{R}, \mathcal{D}, \mathcal{T}, \varphi_{\text{pre}}, \varphi_{\text{inv}}, \varphi_{\text{post}}, I_t, \overline{I_p}, m, \tau_e).$$

Except for a rewriting rule with invariants, the label λ specifies a time interval $I_t \subseteq [0, \infty)$ from which the player can choose the time bound for the rule and, if there are other continuous parameters p_1, \dots, p_n , also an interval $I_{p_j} \subseteq \mathbb{R}$ for each parameter. The element $m \in \{1, *, \infty\}$ specifies if the player must choose a single match of the rule ($m = 1$), apply it simultaneously to all possible matches ($m = \infty$, useful for modeling nature) or if any number of non-intersecting matches might be chosen ($m = *$); τ_e tells which relations must be matched exactly (all other are in τ_h).

We define a *general structure rewriting game* with k players as a directed graph in which each vertex is labelled by k sets of labels denoting possible actions of the players. For each k -tuple of labels, one from each set, there must be an outgoing edge labelled by this tuple, pointing to the next location of the game if these actions are chosen by the players. There can be more than one outgoing edge with the same label in a vertex: In such case, all edges with this label must be assigned probabilities (i.e. positive real numbers which sum up to 1). Moreover, an end-point of an interval I_t or I_p in a label can be given by a parameter, e.g. x . Then, each outgoing edge with this label must be marked by $x \sim \mathcal{N}(\mu, \sigma)$, $x \sim \mathcal{U}(a, b)$ or $x \sim \mathcal{E}(\lambda)$, meaning that x will be drawn from the normal, uniform or exponential distribution (these 3 chosen for convenience). Additionally, in each vertex there are k real-valued terms of the logic presented above which denote the payoff for each player if the game ends at this vertex.

A play of a structure rewriting game starts in a fixed first vertex of the game graph and in a state represented by a given starting structure. All players choose rules, matches and time bounds allowed by the labels of the current vertex such that the tuple of rules can be applied simultaneously. The play proceeds to the next vertex (given by the labeling of the edges) in the changed state (after the application of the rules). If in some vertex and state it is not possible to apply any tuple of rules, either because no match is found or because of the constraints, then the play ends and payoff terms are evaluated giving the outcome for each player.

Example. Let us define tic-tac-toe in our framework. The state of the game is represented by a structure with 9 el-

elements connected by binary row and column relations, R and C , as depicted on the right in Figure 2. To mark the moves of the players we use unary relations P and Q . The allowed move of the first player is to mark any unmarked element with P and the second player can mark with Q . Thus, there are two states in the game graph (representing which player's turn it is) and two corresponding rules, both with one element on each side (left in Figure 2). The two diagonal relations can be defined by $D_1(x, y) = \exists z(R(x, z) \wedge C(z, y))$ and $D_2(x, y) = \exists z(R(x, z) \wedge C(y, z))$ and a line of three by $L(x, y, z) = (R(x, y) \wedge R(y, z)) \vee (C(x, y) \wedge C(y, z)) \vee (D_1(x, y) \wedge D_1(y, z)) \vee (D_2(x, y) \wedge D_2(y, z))$. Using this definitions, the winning condition for the first player is given by $\varphi = \exists x \exists y \exists z (P(x) \wedge P(y) \wedge P(z) \wedge L(x, y, z))$ and for the other player analogously with Q . To ensure that the game ends when one of the players has won, we take as a precondition of each move the negation of the winning condition of the other player.

Playing the Games

When playing a game, players need to decide what their next move is. To represent the preferences of each player, or rather her expectations about the outcome after each step, we use *evaluation games*. Intuitively, an evaluation game is a statistical model used by the player to assess the state after each move and to choose the next action. Formally, an evaluation game \mathcal{E} for \mathcal{G} is just *any* structure rewriting game⁵ with the same number of players and with extended signature. For each relation R and function f used in \mathcal{G} we have two symbols in \mathcal{E} : R and R_{old} , respectively f and f_{old} .

To explain how evaluation games are used, imagine that players made a concurrent move in \mathcal{G} from \mathfrak{A} to \mathfrak{B} in which each player applied his rule $\mathcal{L}_i \rightarrow_{s_i} \mathfrak{R}_i$ to certain matches. We construct a structure \mathfrak{C} representing what happened in the move as follows. The universe of \mathfrak{C} is the universe of \mathfrak{B} and all relations R and functions f are as in \mathfrak{B} . Further, for each $b \in \mathfrak{B}$ let us define the corresponding element $a \in \mathfrak{A}$ as either b , if $b \in \mathfrak{A}$, or as $s_i(b)$, if b was in some right-hand side structure \mathfrak{R}_i and replaced a . The relation R_{old} contains the tuples \bar{b} which replaced some tuple $\bar{a} \in R^{\mathfrak{A}}$. The function $f_{\text{old}}(b)$ is equal to $f_{\text{old}}(a)$ (evaluated in \mathfrak{A}) if b replaced a and it is 0 if b did not replace any element. We use \mathfrak{C} as the starting structure for the evaluation game \mathcal{E} . This game is then played (as described below) and the outcome of \mathcal{E} is used as an assessment of the move \mathfrak{C} for each player.

As you can see above, the evaluation game \mathcal{E} is used to predict the outcomes of the game \mathcal{G} . This can be done in many ways: In one basic case, no player moves in the game \mathcal{E} — there are only probabilistic nodes and thus \mathcal{E} represents just a probabilistic belief about the outcomes. In another basic case, \mathcal{E} returns a single value — this should be used if the player is sure how to assess a state, e.g. if the game ends there. In the next section we will construct evaluation games in which players make only trivial moves depending on certain formulas — in such case \mathcal{E} represents a more complex probability distribution over possible payoffs. In general, \mathcal{E} can be an intricate game representing the judgment process

⁵In fact it is not a single game \mathcal{E} but one for each vertex of \mathcal{G} .

of the player. In particular, note that we can use \mathcal{G} itself for \mathcal{E} , but then without evaluation games any more to avoid circularity. This corresponds to a player simulating the game itself as a method to evaluate a state.

We know how to use an evaluation game \mathcal{E} to get a payoff vector (one for each player) denoting the expected outcome of a move. These predicted outcomes are used to choose the action of player i as follows. We consider all discrete actions of each player and construct a matrix defining a normal-form game in this way. Since we approximate ODEs by polynomials symbolically, we keep the continuous parameters playing \mathcal{E} and get the payoff as a piecewise polynomial function of the parameters. This allows to solve the normal-form game and choose the parameters optimally. To make a decision in this game we use the concept of iterated regret minimization (over pure strategies), well explained in [7].

The regret of an action of one player when the actions of the other players are fixed is the difference between the payoff of this action and the optimal one. A strategy minimizes regret if it minimizes the maximum regret over all tuples of actions of the other players. We iteratively remove all actions which do not minimize regret, for all players, and finally pick one of the remaining actions at random. Note that for turn-based games this corresponds simply to choosing the action which promises the best payoff. In case no evaluation game is given, we simply pick an action randomly and the parameters uniformly, which is the same as described above if the evaluation game \mathcal{E} always gives outcome 0.

With the method to select actions described above we can already play the game \mathcal{G} in the following basic way: Let all players choose an action as described and play it. While we will use this basic strategy extensively, note that, in case of poor evaluation games, playing \mathcal{G} like this would normally result in low payoffs. One way to improve them is the Monte-Carlo method: Play the game in the basic way K times and, from the first actions in these K plays, choose the one that gave the biggest average payoff. Already this simple method improves the play considerably in many cases. To get an even better improvement we simultaneously construct the UCT tree, which keeps track of certain moves and associated confidence bounds during these K plays.

A node in the UCT tree consists of a position in the game \mathcal{G} and a list of payoffs of the plays that went through this position. We denote by $n(v)$ the number of plays that went through v , by $\bar{\mu}(v)$ the vector of average payoffs (for each of the players) and by $\bar{\sigma}(v)$ the vector of square roots of variances, i.e. $\sigma_i = \sqrt{\sum_{p_i} (p_i^2)/n - \mu_i^2}$ if p_i are the recorded payoffs for player i . First, the UCT tree has just one node, the current position, with an empty set of payoffs. For each of the next K iterations the construction of the tree proceeds as follows. We start a new play from the root of the tree. If we are in an internal node v in the tree, i.e. in one which already has children, then we play a regret minimizing strategy (as discussed above) in a normal-form game with payoff matrix given by the vectors $\bar{\mu}'(w)$ defined as follows. Let $\sigma'_i(v) = \sigma_i(v)^2 + \Delta \cdot \sqrt{\frac{2 \ln(n(v)+1)}{n(w)+1}}$ be the upper confidence bound on variance and to scale it let $s_i(v) =$

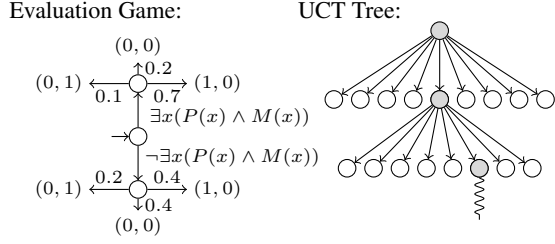


Figure 3: Evaluation game for tic-tac-toe and a UCT tree.

$\min(1/4, \sigma'_i(v)/\Delta)$, where Δ denotes the payoff range, i.e. the difference between maximum and minimum possible payoff. We set $\mu'_i(w) = \mu_i(w) + C \cdot \Delta \cdot \sqrt{\frac{\ln(n(v)+1)}{n(w)+1}} s_i(v)$. The parameter C balances exploration and exploitation and the thesis [3] gives excellent motivation for precisely this formula (UCB1-TUNED). Note that for turn-based games, when player i moves, we select the child w which maximizes $\mu'_i(w)$. When we arrive in a leaf of the UCT tree, we first add all possible moves as its children and play the evaluation game a few ($E > 1$) times in each of them. The initial value of $\bar{\mu}$ and $\bar{\sigma}$ is computed from these evaluation plays (both must be set even if $n = 0$). After the children are added, we select one and continue to play with the very basic strategy: Only the evaluation game is used to choose actions and the UCT tree is not extended any more in this iteration. When this play of \mathcal{G} is finished, we add the received payoff to the list of recorded payoffs of each node on the played path and recalculate $\bar{\mu}$ and $\bar{\sigma}$. Observe that in each of the K iterations exactly one leaf of the UCT tree is extended and all possible moves from there are added. After the K -th iteration is finished, the action in the root of the UCT tree is chosen taking into account only the values $\bar{\mu}$ of its children.

Example. Consider the model of tic-tac-toe presented previously and let the formula $M(x) = \exists y C(x, y) \wedge \exists y C(y, x) \wedge \exists y R(x, y) \wedge \exists y R(y, x)$ express that x is the position in the middle of the board. In Figure 3 we depicted a simple evaluation game, which should be interpreted as follows. If the first player made a move to the middle position, expressed by $\exists x(P(x) \wedge M(x))$, then the probability that the first player will win, i.e. of payoff vector $(1, 0)$, is 0.7. The probability that the second player will win is 0.1 and a draw occurs with probability 0.2. On the other hand, if the first player did not move to the middle, then the respective probabilities are 0.4, 0.2 and 0.4. When the construction of the UCT tree starts, a payoff vector is assigned to the state after each of the 9 possible moves of the first player. The payoff vector is one of $(1, 0)$, $(0, 1)$ and $(0, 0)$ and is chosen randomly with probabilities 0.7, 0.1, 0.2 for the middle node in the UCT tree and with probabilities 0.4, 0.2, 0.4 for all other 8 nodes, as prescribed by the evaluation game. The first iteration does not expand the UCT tree any further. In the second iteration, if the middle node is chosen to play, then its 8 children will be added to the UCT tree. The play in this iteration continues from one of those children, as depicted by the snaked line in Figure 3.

Learning Evaluation Games

Even during a single play of a game \mathcal{G} we construct many UCT trees, one for each move of each player, as described above. A node appearing in one of those trees represents a state of \mathcal{G} and a record of the payoffs received in plays from this node. After each move in \mathcal{G} , we collect nodes from the UCT tree from which a substantial number of plays were played, i.e. which have n bigger than a confidence threshold N . These nodes, together with their payoff statistics, are used as a training set for the learning procedure.

The task of the learning mechanism started on a training set is to construct an evaluation game \mathcal{E} , preferably as simple as possible, which, started in a state from the training set, gives payoffs with a distribution similar to the one known for that state. Observe that the game \mathcal{E} constructed for a training set from some plays of \mathcal{G} is therefore a simplified probabilistic model of the events which occurred during simulated plays of \mathcal{G} . Further — a game \mathcal{E} constructed from plays of \mathcal{G} which already used an evaluation game \mathcal{E}' models plays between players who already “know” \mathcal{E}' . Note how this allows incremental learning: each evaluation game can be used to learn a new one, modeling plays between smarter players.

We present a non-deterministic construction of candidate evaluation games, i.e. we provide only the basic operations which can be used and leave the choice to an external function. Still, as we show next, even very simple choices can produce useful evaluation games. During the whole construction process the procedure maintains a few sets: a set G of evaluation games, a set S of structures, a set T of equational terms, and a set Φ of formulas and real-valued terms. Initially the set G contains at least a trivial game, S a trivial structure and T and Φ may be empty. The learning procedure constructs new games, structures, terms and formulas until, at some point, it selects one game from G as the result.

The rules for adding new formulas and terms to Φ closely resemble the syntax of our logic presented before. For each syntactic rule we allow to add to Φ its result if the required formulas and terms are already in Φ . For example, we can add the literal $P(x)$ to an empty set Φ , then add $Q(x)$, create $P(x) \wedge Q(x)$, and finally use the existential quantifier to create $\exists x(P(x) \wedge Q(x))$. The rules for construction of equational terms are analogous. For structures, we allow to add a single element or a single relation tuple to a structure from S and to take disjoint sum of two structures from S . Finally for games we allow compositional rules similar to the constructions in Parikh’s Game Logic, cf. [10].

Clearly, the rules above are very general and it is up to the external function to use them to create a good evaluation game. In our first experiments, we decided to focus on a very simple heuristic which does not create any structures or equations. It uses only formulas and probabilistic vertices and the payoff is always one of the vectors already occurring in the training set. Moreover, we do not allow arbitrary formulas but only existentially quantified conjunctions of literals. Evaluation games created by our function have thus similar form to the one presented on the left side of Figure 3. To decide which formula to add next, our function extends formulas already kept in Φ by a literal and keeps the one which is the best selector. This is very similar to the

	White uses \mathcal{E}	Black uses \mathcal{E}
Gomoku	78%	82%
Breakthrough	77%	73%

Table 1: Playing with a learnt evaluation against pure UCT.

Apriori algorithm for frequent itemset mining, just instead of items we use literals and a set of literals is understood as their conjunction, existentially quantified. Transactions in this sense are sets of states with average payoff in a specific interval. The found formulas are then used as constraints of a transition to a probabilistic node which is constructed so as to model the distribution of payoffs in the states from the training set which satisfy this formula. (cf. Figure 3).

Experimental Results

The described algorithms are a part of Toss (sourceforge.net), an open-source program implementing the presented game model and a GUI for the players.⁶ To construct a scalable solver for our logic we used a SAT solver (MiniSAT) to operate on symbolic representations of MSO variables and implemented a quantifier elimination procedure for real numbers based on Muchnik’s proof. The UCT algorithm and the rewriting engine were implemented in OCaml and the GUI in Python using the Qt4 library. In Toss, we defined a few board games and some systems with continuous dynamics. In this paper, we present preliminary results for Breakthrough and Gomoku, two games often used to evaluate game playing programs.

The strength of the UCT algorithm has been evaluated before: it is respectable, but can only go so far without any evaluation function. We used our learning procedure to get an evaluation game for both Breakthrough and Gomoku. Only top-performing formulas were selected, in case of Breakthrough it was one simple formula meaning “beat if possible” and for Gomoku the formulas suggested to put stones near the already placed ones. While these formulas are basic, we present in Table 1 the percentage of plays won by a UCT player using the evaluation game against an opponent using none. As you can see, this is a significant majority of cases — even playing black in Breakthrough and white in Gomoku, i.e. starting second, which is a weaker position.

Perspectives

We presented a general model of games and an algorithm which can both play the game in a reasonable way and learn from past plays. There are several phenomena which we did not include in our model. On the side of games, we allowed neither imperfect nor incomplete information, so players must fully know the game and its state, which is not realistic. On the modeling side, relational structures give no direct way to represent hierarchies, which should be improved as well. For practical use of our system on larger examples it is also important to introduce a type system, which

⁶Currently released version 0.3 of Toss does not support all the features we described. Our tests were conducted on a recent version in code repository and will be included in the next release.

should be integrated with our logic. We started to investigate this problem from the theoretical side in [8]. To improve the learning procedure, we plan to investigate classical learning algorithms (e.g. C4.5), and recent program induction methods (e.g. [9]). These can hopefully find new rewriting rules and in this way generate novel evaluation games. One could also try to analyze UCT using higher-order probabilities [6].

Even with the drawbacks mentioned above, the model we presented is, to our knowledge, the most general kind of games for which a playing algorithm is implemented. In addition to this generality, we have chosen a hypergraph representation for states, which is already used in AGI projects. Our playing algorithm is based on the upper confidence bounds method, which is not only established for board games but also scales to other domains, e.g. robotic visual learning [11]. Thus, it could be opportune to use our system as a basis for an AGI project and we look forward to cooperating with AGI system builders on such integration.

References

- [1] S. Burmester, H. Giese, E. Münch, O. Oberschelp, F. Klein, and P. Scheideler. Tool support for the design of self-optimizing mechatronic multi-agent systems. *International Journal on Software Tools for Technology Transfer*, 10(3):207–222, 6 2008.
- [2] H. Finnsson and Y. Björnsson. Simulation-based approach to general game playing. In *Proc. of AAAI’08*. AAAI Press, 2008.
- [3] S. Gelly. *A Contribution to Reinforcement Learning; Application to Computer-Go*. Dissertation, University Paris-Sud 11, 2007.
- [4] M. R. Genesereth, N. Love, and B. Pell. General game playing: Overview of the AAAI competition. *AI Magazine*, 26(2):62–72, 2005.
- [5] B. Goertzel. Patterns, hypergraphs and embodied general intelligence. In *Proc. of IJCNN’06*, pages 451–458, 2006.
- [6] B. Goertzel, M. Ikle, and I. L. Freire Goertzel. *Probabilistic Logic Networks: A Comprehensive Framework for Uncertain Inference*. Springer, 2008.
- [7] J. Y. Halpern and R. Pass. Iterated regret minimization: A new solution concept. In *Proc. of IJCAI’09*, pages 153–158, 2009.
- [8] Ł. Kaiser. Synthesis for structure rewriting systems. In *Proc. of MFCS’09*, volume 5734 of *LNCS*, pages 415–427. Springer, 2009.
- [9] M. Looks and B. Goertzel. Program representation for general intelligence. In *Proc. of AGI’09*, 2009.
- [10] R. Ramanujam and S. Simon. Dynamic logic on games with structured strategies. In *Proc. of KR’08*, pages 49–58. AAAI Press, 2008.
- [11] M. Salganicoff, L. H. Ungar, and R. Bajcsy. Active learning for vision-based robot grasping. *Machine Learning*, 23:251–278, 1996.

Neuroethological Approach to Understanding Intelligence

DaeEun Kim

Biological Cybernetics Lab
School of Electrical and Electronic Engineering
Yonsei University
Seoul, 120-749, South Korea

Abstract

The neuroethology is an interdisciplinary study among artificial intelligence, biology and robotics to understand the animal behavior and its underlying neural mechanism. We argue that the neuroethological approach helps understand the general artificial intelligence.

Introduction

Many animal behaviours demonstrate unsolved mysteries and some biologists have studied to answer the question of how their intelligence is connected to the behaviours. Their intelligence level is not superior to human intelligence, but the study of animal intelligence provides a clue to understand what kind of neural mechanism supports the intelligence. This study stimulates relevant studies in the field of robotics and artificial intelligence. Robotists have worked to find how sensory stimuli are connected to motor actions, what is a good mapping from sensors to motors, how the learning mechanism helps to build more complete information about the environment, and what is the optimal control based on the acquired environmental information. All these kinds of problems can also be found in the analysis of animal behaviours as well as human brain system. Still unknown are their underlying mechanisms in details.

Some scientists simulate human brain models to understand the general intelligence, and analyze a map of neurons in terms of the function of human brain, and try to answer how the system of neurons works. Assuming all biological animals have diverged from their ancestral animals with respect to the evolutionary track, we can guess that their brain system and the functions should have similar neural mechanisms and operations. The neuroethological approach explains how and why the animals behave at a specific situation or survive their environments. It shows how their brain system organizes the sensory-motor loop and issues appropriate commands of control for desirable behaviours, and ultimately may explain what is the intelligence. From invertebrate animals to humans, the intelligence follows evolutionary steps as the complexity of the brain system develops. We believe that this kind of study can

demonstrate all the levels of intelligence, and furthermore, elements of artificial general intelligence.

Neuroethological Approach

The neuroethology, as a multidisciplinary study between neurobiology and ethology, focuses on the study of interaction of neural circuitry to explain the behavioural concepts or strategies found in animals. This field covers the underlying mechanism and analysis to cover high-level features of animal cognition. There is an example of neuroethological approach to understand a part of the artificial general intelligence. Passive sensing and active sensing depending on motor actions are classified as a different level of intelligence. Passive sensing implies receiving sensory information from the environment without any sensor-motor coordination. In contrast, active sensing is involved with a sequence of motor actions which change the environment or activate the body movement, and ultimately produces another view of sensory information.

A collection of sensory information depending on a sequence of motor actions provides information of the environment in more details. For example, bats have an echolocation system to search for food. They use the ultrasonic system to emit the signal and sense the reflected sonar signal to measure the location, size and the moving speed of their prey. Similarly, electric fish have electrolocation by generating the electric field and observe the change of the electric field distorted by the surrounding objects. These animals have a relatively large size of the brain to process the active sensing results. It is believed that their brain systems have a prediction model for their self-generated motor actions to estimate accurately the location of their target object. This system corresponds to a forward model in the human brain system. The cerebellum includes the forward model and inverse model for their motor actions and sensory information (WK98). Also, it is presumed that crickets show a low-level forward model for auditory system (Web04). From the neuroethological analysis, we can infer the fundamental mechanism of the brain system and thus a predictive system for intelligence can be explained.

Biologists and neuroethologists have studied sensory

systems and their sensorimotor coordination (Car00). Bats use echolocation for prey capture and navigation in the cave, and electric fish show underwater navigation and electric communication. Barn owls have their specialized auditory system. Aplysia show the fundamental mechanism of learning and memory with a small number of neurons. Rats use spatial cognition and memory, and navigate in the dark. Crabs show visual information processing, the eye-leg coordination and path integration (ZH06). Honeybees have visual navigation, learning and its flight mechanism depending on vision (SZ00). Many animals have their own features of sensory mechanism, the coordination of motor actions, or adaptability and learning mechanism, and many of them can partly provide a key to understand the intelligence.

Animals have robust mechanism adapting themselves to their environment. Their sensory processing and pattern recognition are organized to extract relevant information from the environmental system. The system robustly works even in noisy environments. The motor actions triggered by sensory information are regulated efficiently and a sequence of multiple motor actions are integrated in high skills. This kind of dexterity should be explained by neural mechanism to understand the intelligence.

Robotics

There have been many robotic applications that look intelligent to the public. However, in many cases, the approach is far from the artificial general intelligence. Robotic problems consider a specific application and they are often based on engineering analysis and design. Intelligence is a side-effect with the approach, but neurorobotics and neuroethological approach to robotics have demonstrated the neural mechanism and its application motivated by biological systems. It is a part of systematic and integrative approach to the intelligence. The current state of the work still has a narrow range of scope for the artificial general intelligence. If more integrative methods are available to explain high-level features of intelligence such as planning and inference, it would help understand the general artificial intelligence.

Intelligent robots handle how a robotic agent behaves autonomously. The robotic agent behaviours resemble the animal behaviours with respect to the system model including sensory system, motor actions, sensory-motor coordination, and further, a hierarchy of the interaction levels among the system elements. Thus, understanding animal behaviours may lead to a possible application of intelligent robotic system. In addition, understanding human brain system can produce intelligent machines with human-level cognition.

Artificial General Intelligence

The general intelligence needs to cover all levels of intelligence from low to high level of features. How the

system is organized into a high level of features is still an open question. Another program or concept is required for the artificial general intelligence. How the agent generalizes the knowledge over various domains or modalities and how the agent conceptualize the instances are interesting problems in this field. The process should be modelled with a neuronal system and it is believed that it requires a complex type of adaptation or networked organization. We expect the features could be explained by complex networks over a collection of neuron units, where each neuron has its own adaptivity and learning. A simple neuron itself has adaptation for the synaptic plasticity depending on their inputs and output. This kind of adaptation may lead to a complex structure of agents and explain how agents interact each other. It seems it may be related to emergent properties of a collection of units.

This neuroethological study may reveal the basic element of the artificial general intelligence for learning and adaptation process. Especially it has a potential of application for robotic agents to behave intelligently as natural animals do. Yet we still need further work to relate the neuroethology to the high-level agent architecture including reasoning, planning and solving problems.

Conclusion

We suggest a neuroethological approach to robotics can explain the behaviour-level intelligence and its underlying neural mechanism. It will explain many important concepts of sensory integration, sensor-motor coordination and learning mechanism needed for the general artificial intelligence.

Acknowledgments

This work was supported by the Korea Science and Engineering Foundation(KOSEF) grant funded by the Korea government(MEST) (No. 2009-0080661)

References

- T.J. Carew. *Behavioral Neurobiology*. Sinauer Associates, 2000.
- M.V. Srinivasan and S. Zhang. Honeybee navigation: Nature and calibration of the odometer. *Science*, 287:851–853, 2000.
- B. Webb. Neural mechanisms for prediction: do insects have forward models? *Trends in Neurosciences*, 27:278–282, 2004.
- D. W. Wolpert and M. Kawato. Multiple paired forward and inverse models for motor control. *Neural Networks*, 11:1317–1329, 1998.
- J. Zeil and J.M. Hemmi. The visual ecology of fiddler crabs. *J. of Comparative Physiology A*, 192:1–25, 2006.

The Toy Box Problem (and a Preliminary Solution)

Benjamin Johnston¹

Business Information Systems
Faculty of Economics and Business
The University of Sydney
NSW 2006, Australia

Abstract

The evaluation of incremental progress towards ‘Strong AI’ or ‘AGI’ remains a challenging open problem. In this paper, we draw inspiration from benchmarks used in artificial commonsense reasoning to propose a new benchmark problem—the Toy Box Problem—that tests the practical real-world intelligence and learning capabilities of an agent. An important aspect of a benchmark is that it is realistic and plausibly achievable; as such, we outline a preliminary solution based on the Comirit Framework.

Introduction

The objective of commonsense reasoning is to give software and robotic systems the broad every-day knowledge and know-how that comes effortlessly to human beings and that is essential for survival in our complex environment. While commonsense is, or at least appears to be, a narrower problem than ‘Strong AI’ or ‘AGI’, it shares many of the same representational, computational and philosophical challenges. Indeed, one might view commonsense as the practical, immediate and situated subset of general purpose intelligence.

Thus, commonsense reasoning serves as a useful stepping-stone towards both the *theory* and *practice* of Strong AI. Not only does commonsense provide a broad, deep and accessible domain for developing *theoretical* conceptions of real-world reasoning, but the *practical* experience of developing large scale commonsense provides useful insights into the scalability challenges of general-purpose intelligence.

In much the same way that evaluation challenges AGI researchers today², work in commonsense reasoning has long been challenged by the difficulty of finding manageable, but open-ended benchmarks.

The difficulty of evaluation facing both commonsense reasoning and AGI arises from the fact that the problems are so great that we cannot today simply implement *complete* systems and test them in their intended domains; it is necessary to measure *incremental* progress. Unfortunately, incremental progress can be difficult to judge: it is relatively easy to demonstrate that a particular formalism supports or lacks a given feature, but it is much harder to determine whether that feature represents a meaningful improvement. For example, even though a formalism or system appears to offer

improvements in expressive power, efficiency and ease-of-use, it may have sacrificed a ‘show-stopping’ crucial feature that is overlooked in the evaluation.

In the commonsense reasoning community, this problem is addressed by defining non-trivial (but plausibly achievable) reasoning problems, and then analyzing the ability of a formalism to solve the problem and a number of elaborations. While analysis is conducted with respect to the system’s performance (rather than ‘features’), the formalism itself is also considered in what might be termed a ‘grey box analysis’ (rather than, say, a black-box comparison such as used at RoboCup). To these ends, Morgenstern and Miller (2009) have collected a set of non-trivial commonsense reasoning problems and a handful of proposed ‘solutions’.

With any benchmark or challenge problem there is, of course, the temptation to fine-tune a system to the problem, rather than attempting to design more general and abstract capabilities. As such, the benchmarks used in the commonsense reasoning community are not formally defined, nor are they strictly measurable. Instead, they offer a shared and open-ended scenario to guide qualitative analysis of progress towards our goals. Even though this approach is less objective than a competition or a formal goal, these challenge problems provide a meaningful context for evaluation that helps temper unfounded wild claims, while at the same time avoiding specifics that are readily gamed.

To date, we have been developing a general purpose commonsense-reasoning framework named *Comirit*, and have evaluated the system on two benchmark problems: the Egg Cracking Problem (Johnston and Williams 2007), and the Eating on an Aircraft Problem (Johnston 2009). We have found this methodology useful, but in charting a course to more general intelligence, we found that the current selection of proposed benchmark problems offer little scope for evaluating the ability of an agent to learn on its own and thus demonstrate AGI-like capabilities.

Our objective in this paper is therefore to present an open-ended benchmark, the *Toy Box Problem*, in the style of Morgenstern and Miller (2009), which may be used to evaluate progress towards commonsense reasoning and general intelligence.

In order to briefly illustrate the feasibility of the benchmark problem and the problem may be applied, we then use the *Toy Box Problem* on the Comirit framework. In doing so, we will introduce new capabilities in the framework, and show how the framework may be used to partially solve the Toy Box Problem.

¹ This research supported in part by an ARC Discovery Grant while at the University of Technology, Sydney.

² Consider, for example, the ‘Developing an AGI IQ Test’ workshop that is affiliated with the AGI 2010 conference.

The Toy Box Problem

As with existing benchmarks used within the commonsense reasoning community (Morgenstern and Miller 2009), we pose the *Toy Box Problem* as a hypothetical scenario:

A robot is given a box of previously unseen toys. The toys vary in shape, appearance and construction materials. Some toys may be entirely unique, some toys may be identical, and yet other toys may share certain characteristics (such as shape or construction materials). The robot has an opportunity to first play and experiment with the toys, but is subsequently tested on its knowledge of the toys. It must predict the responses of new interactions with toys, and the likely behavior of previously unseen toys made from similar materials or of similar shape or appearance. Furthermore, should the toy box be emptied onto the floor, it must also be able to generate an appropriate sequence of actions to return the toys to the box without causing damage to any toys (or itself).

The problem is intentionally phrased as a somewhat constrained (but non-trivial) scenario with open-ended possibilities for increasing (or decreasing) its complexity. In particular, we allow the problem to be instantiated in combinations of four steps of increasing situation complexity and four steps of toy complexity.

That is, the problem may be considered in terms of one of the following environments:

- E1. A virtual robot interacting within a virtual 2-dimensional world
- E2. A real robot interacting within a real-world planar environment (*e.g.*, a table surface with ‘flat’ toys and in which no relevant behavior occurs above the table surface)
- E3. A virtual robot interacting within a virtual 3-dimensional world
- E4. A real robot interacting within the real world, without constraints

Similarly, the complexity of toys is themselves also chosen from the following:

- T1. Toys with observable simple structure, formed from rigid solids, soft solids, liquids and gases
- T2. Toys with complex, but observable mechanical structure (again, created from rigid solids, soft solids, liquids and gases)
- T3. Toys with complex, but observable mechanical structure, created from any material (including magnets, gases and reactive chemicals)
- T4. Toys with arbitrary structure and operation (including electronic devices)

In each case, the world and the toys contained within, may only be observed via ‘raster’ cameras. That is, even in virtual worlds (E1 and E3), the robot is unable to directly sense the type or the underlying model of a virtual toy (*i.e.*, the problem cannot be tested in a world such as Second Life, in which agents can directly ‘sense’ the symbolic name, type and properties of an object).

In fact, virtual worlds (of E1 and E3) should be as close as possible to a physical world (including the ability for objects to be arbitrarily broken and joined). Virtual worlds are included in the Toy Box Problem not to reduce the conceptual

challenge of the problem, but primarily to separate the effort involved in dealing with camera noise, camera/hardware failures, color blurring and other sensory uncertainty.

The Toy Box Problem may be used for evaluating an ‘intelligent’ system by selecting a combination of environment and toy challenges. For example, the pairing E1&T1 represents the easiest challenge, whereas E4&T4 present the greatest difficulty. Note, however that the pairs do not need to match: the next development step for a system which solves E1&T1 might be either E1&T2 or E2&T1.

The Toy Box Problem is specifically designed as a *stepping stone* towards general intelligence. As such, a solution to the simplest instances of this problem should not require universal or human-like intelligence. While an agent must have an ability to learn or identify by observation (because the toys are new to the agent), it does not *necessarily* require the ability to ‘learn to learn’. For example, given a comprehensive innate knowledge-base of naïve physics, it *may* be sufficient for an agent to solve the problem with toys in T1 and T2 by a process of identification rather than true learning. However, the difficulty of the challenge increases with more complex toys of T3 and T4, and it is unlikely that a system would continue to succeed on these challenges without deeper learning capabilities (though, it would be a very interesting outcome with deep implications for AGI research if a system without learning capabilities does continue to succeed even on the most challenging instances of the problem).

While the pairing E1&T1 is the easiest challenge of the Toy Box Problem, we believe that any solution to E1&T1 would be a non-trivial accomplishment, far beyond the reach of standard ‘Narrow AI’ techniques in use today. Nevertheless, we expect that the pair E1&T1 should lie within reasonable *expectations* of the capabilities of proposals for ‘Strong AI’ architectures today. One could readily conceive of systems based on methods as diverse as logic, connectionist networks or genetic programming to each be adapted to solving E1&T1 within a short-term project, and thus form the basis of meaningful comparison and analysis between disparate methods.

More difficult combinations, such as the pair E4&T4, are currently far beyond all current technologies. While a system that performs well for such pairings may not have true general intelligence, it would be at the pinnacle of practical real-world physical competence and would have serious real world applications. For example, this level of knowledge would enable a domestic robot or a rescue robot to deal with the many unexpected objects and situations it would encounter during its routine duties: whether cleaning a house or making way through urban rubble.

Finally, it is worthwhile noting a connection here with recent discussions concerning the creation of virtual ‘pre-schools’ for human-like AI development (Goertzel and Bugaj 2009). The Toy Box Problem may be viewed as a specific and achievable ‘target’ for developing and evaluating real systems, rather than simply aiming to provide an enriching environment for robot ‘education’.

In this rest of this paper, we further illustrate the problem by outlining a preliminary solution to the Toy Box Problem, and considering (in the first instance) how it may be ‘solved’ for the pair E1&T1.

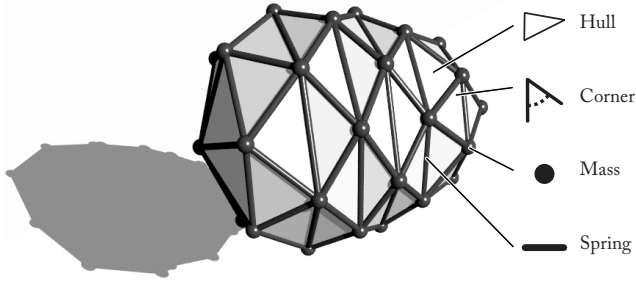


Figure 1: Parts of a simulation of an egg

Comirit Framework

Over the past four years, we have been developing Comirit; a hybrid reasoning framework for commonsense reasoning. At the core of the framework lies a generic graph-based scheme for constructing simulations that implicitly capture practical commonsense knowledge. While the framework is not intended to be biologically plausible, simulation in Comirit may be viewed as a computational analog to human visual imagination. Comirit uses simulations to reason about, predict and speculate about a given situation, by first instantiating a simulation of that situation and then using the simulation as a mental playground for experimenting with possible actions and anticipating reactions.

However, while simulation is a powerful, computationally efficient, and easy-to-engineer scheme for representing commonsense knowledge and predicting the outcome of a situation, the method is constrained to concrete reasoning and to the ‘arrow-of-time’. That is, simulation by itself is not well suited to the following kinds of reasoning:

1. Explaining the cause of an outcome (‘Why is there split milk on the floor?’)
2. Fact-based reasoning (‘What is the capital of Russia?’)
3. Abstract deduction (‘What is 3 plus 7?’)
4. Learning about and predicting in novel domains (‘How will this new toy behave?’)

We have therefore developed Comirit as an open-ended multi-representational framework that combines simulation with logical deduction, machine learning and action selection. This integration is achieved by a uniform mechanism that is based on the automated theorem proving method of analytic tableaux (see *e.g.*, Hähnle 2001). In Comirit, the tableau algorithm is extended so that it searches and ranks spaces of possible worlds, enabling the disparate mechanisms to be uniformly represented and reasoned in a unified tableau.

In the remainder of this section, we provide an overview of simulation, hybrid reasoning and learning in Comirit and show how it relates to the *Toy Box Problem*. More detailed explanations of Comirit may be found in our prior publications (Johnston and Williams 2007; 2008; 2009).

Simulation

In the Comirit framework, simulations are the primary representation of commonsense knowledge. Comirit Simulations are designed as a generalization and formalization of an early proposal by Gardin and Meltzer (1989). In particular, Comirit

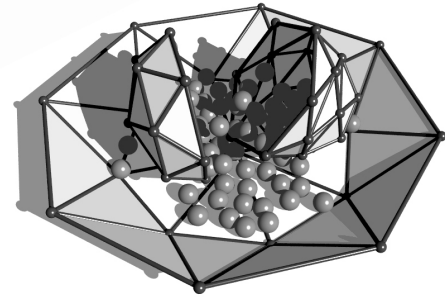


Figure 2: Simulation of an egg cracked into a bowl (the beads represent spilled yolk and white)

uses a generic graph-based representation that has been extended to use accurate 3D physics³.

Complex objects are modeled by approximating the structure of the object as an annotated graphical structure, and then iteratively updating the annotations according to the laws of physics. That is, if we have an object to simulate—an egg, for example—then a graph is instantiated comprising of vertices that denote interconnected ‘masses’, ‘springs’, ‘torsion springs’ and ‘convex hulls’ which approximate the structure of an egg. Each such vertex is annotated with attributes to drive appropriate behavior; for example, each ‘mass’ vertex has a spatial position (x, y, z coordinates) and a local mass density (among other attributes). Newton’s laws of motion are then iteratively applied to the graph structure. For example, the effects of springs, liquids and hulls are modeled using Hooke’s law. Figure 1 illustrates some of the parts of a simulation for the Egg Cracking Problem. Figure 2 also depicts the result of running such a simulation to observe the effects of dropping an egg into a bowl: the egg has cracked, and its liquid contents have spilled out.

Comirit uses 3D simulations and so is potentially applicable to E3 and E4 in the Toy Box Problem, however the same framework may be trivially adapted to the 2D environments of E1 and E2. The translation of 3D physics into 2D physics is straightforward: one axis is either fixed to be constant, or is entirely removed from the equations of motion.

A system may perform powerful reasoning about any object for which it has a simulation. For example, it may consider safe ways of handling eggs and toys by instantiating a simulation in internal memory and then testing actions against that ‘imagined’ instance. If the agent uses visualization to determine that a heavy force will cause an egg to crack, it can avoid causing damage to the egg in real life.

Simulation + Reasoning

Recalling that simulation only supports a ‘forward chaining’ inference mode, we have integrated simulation with logical deduction in a hybrid architecture in order to combine the strengths and complement the weaknesses of each mechanism. That is, we use the deductive power of a general-purpose logic to make up for the inflexibility of simulation.

In combining simulation and logic, our experiences are that

³ Comirit may also support non-physical domains, such as financial markets or organizational behavior but they are beyond the scope of this paper.

the conceptual mismatch between the mechanisms of simulation and logic prevents the application of traditional integration techniques such as blackboard architectures. Our attempts to use mainstream integration architectures invariably resulted in systems that were unworkably complex and difficult to maintain. Instead, we sought a clear and unifying abstraction to harmonize the semantics of the reasoning mechanisms; by interpreting both simulation and logical deduction as operations that manipulate spaces of possible worlds.

The method of analytic tableaux (see *e.g.*, Hähnle 2001) is an efficient method of automatic theorem proving. Analytic tableaux have been successfully applied to large problems on the semantic web, and there is a vast body of literature on their efficient implementation (*ibid.*). The method involves the automatic construction of search trees (tableaux) through the syntactic decomposition of logical expressions, and then eliminates branches of the tree that contain contradictions among decomposed atomic formulae. Each branch of the resultant tableau may be seen as a partial, disjunction-free description of a model for the input formulae.

Logical deduction *and* simulation can be unified through tableau reasoning. The tableau algorithm is designed for logical deduction, and its algorithm is effectively a search through symbolically-defined spaces of worlds. Simulation is a process that can be used to expand upon symbolic knowledge in a given world (*i.e.*, by forward chaining to future states based on description of the current state), and so simulation can be applied to generate information in the branches of a tableau.

Comirit thereby incorporates a generalization of the tableau method such that a tableau may contain not only standard logical terms and formulas, but also non-logical structures such as simulations, functions, data-structures and arbitrary computer code. With some similarity to the methods of Poly-Scheme (Cassimatis 2005), integration in Comirit is achieved by translating diverse reasoning mechanisms into the tableau operators for expansion, branching and closing of branches. Traditional logical tableau rules are used unchanged, and simulation is treated as an expansion operator (like the conjunction rule).

More detailed explanation of the workings of Comirit tableau reasoning (including an explanation of how tableau rules, heuristics and meta-rules are also recursively embedded inside the tableau) may be found in our earlier publication (Johnston and Williams 2008). In the following subsection, we will provide an example of a tableau as it is further extended and used for machine learning.

Simulation + Reasoning + Learning + Action

Of course, even with a comprehensive knowledge base, an intelligent system will be of limited use in any complex and changing environment if it is unable to learn and adapt. Indeed, in the Toy Box Problem, the agent has no prior knowledge of the specific toys that it may encounter. The system must autonomously acquire knowledge through interaction and observation of the toys.

It turns out that simulation is ideal for observation-based learning. The laws of physics are generally constant and universal; an agent does not need to learn the underlying laws of behavior of every object. Thus, when the underlying graphical

structure of an object can be approximated by direct observation, the learning problem is then reduced to discovering the hidden *parameters* of the object by machine learning.

For example, given a novel toy (*e.g.*, a toy ring), direct observation may be used to directly instantiate a graph-based mesh that approximates the structure. In the 3D case, this would be achieved by using the 3D models extracted from stereographic cameras, laser scanners or time-of-flight cameras; in the 2D case, this might be achieved by simple image segmentation.

Once the shape of an object has been approximated by a graph, machine learning is used to determine underlying values of annotations: mass densities, spring constants, rigidity and breaking points that will result in an accurate simulation. These can be discovered simply by collecting observations, and using these observations as training instances for a parameter search algorithm (where fitness is measured by the accuracy of the simulation given the parameters).

However, while simulation is well suited to learning, it is not necessarily obvious how to reconcile the search for consistency that is fundamental to the tableau method with the hypotheses search and evaluation of learning. A given hypothesis can not be independently declared either true or false (as demanded by tableau reasoning): it is only possible to compare hypotheses against each other and select the ‘best’.

Thus, in Comirit, learning is implemented by further extending the tableau reasoning algorithm. Learning in Comirit is treated as a ‘generate-and-test’ algorithm. Generation of candidate hypotheses is akin to the disjunctive branching of the tableau, however the testing of hypotheses is implemented as a special extension of the tableau algorithm to allow branches to be closed due to sub-optimality.

Learning is therefore implemented by introducing an ordering over branches, and then treating the tableau algorithm as a search for both consistent *and* minimal models. A branch in a tableau is no longer advanced or refined (as though ‘open’) simply if it is consistent per the traditional tableaux algorithm: it must be consistent and *have no other consistent branch that compares less in the partial order*. A consistent but non-minimal branch is therefore said to be ‘weakly closed’.

We define the ordering over branches using symbols that are stored *within* the tableau. The set of propositions *rank(Index, Value)* are assumed to be tautologically true in any logical context, but are used for evaluating the order of the branches. The *Index* is an integer indicating the order in which the *Values* are to be sorted: branches are first compared by the values with smallest indexes of any rank term in the branch; equal values are compared using the next smallest rank term; and so on⁴.

To illustrate this process, consider a robot upon encountering a novel toy ball. Using just a naïve stochastic hill-climbing strategy⁵ for generating hypotheses, it may use observations of the object in order to build an accurate simulation of the

⁴ This extension does not affect the consistency or completeness of logical deduction in the framework; the rank terms simply prioritize the search towards minimal branches.

⁵ This strategy is effective in our example, but in real-world settings, more powerful algorithms may be used.

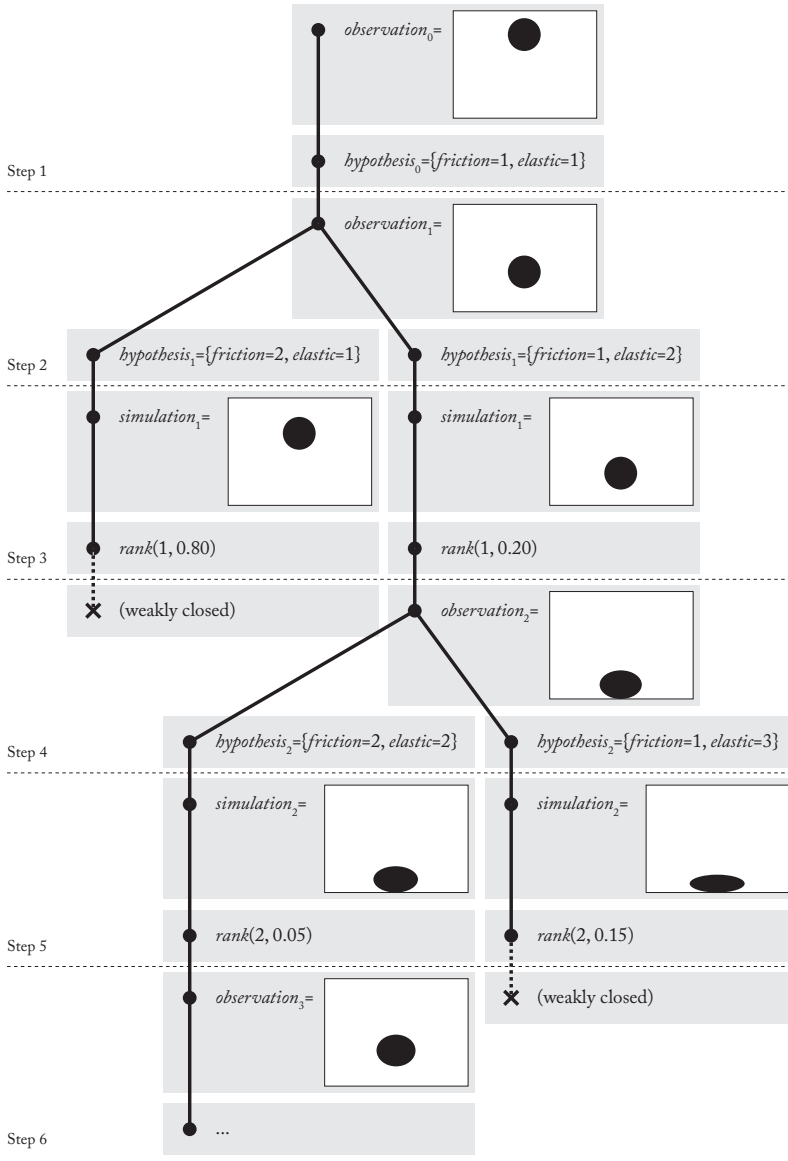


Figure 3: Learning in a tableau

object, as depicted in Figure 3:

- Step 1.** The tableau initially contains the first observation of a ball and the initial hypothesis generated (many other control objects, meshes, functions and other data will be in the tableau, but these are not shown for simplicity).
- Step 2.** The system observes movement in the ball. It generates new hypotheses, seeking to find a hypothesis with minimal error.
- Step 3.** The system simulates each hypothesis. The result of the simulation is compared with observations to determine the error in the hypothesis. The right branch has smaller error so the left branch is ‘weakly closed’.
- Step 4.** As with Step 2, the system observes more movement and generates new hypotheses, further refining the current hypothesis.
- Step 5.** The system then simulates as with Step 3, but this time the left branch is minimal.

Step 6 and later. The algorithm continues yet again with more new observations and further hypothesizing.

Note also that because learning occurs in an (extended) tableau containing logic, simulations and arbitrary functions, the system may use logical constraints or ad-hoc ‘helper functions’ even when searching for values in a simulation (e.g., it may use constraint such as $mass > 0$, or a heuristic-driven hypothesis generator to produce better hypotheses faster).

Furthermore, the ordering induced by rank terms finds application not only in driving the search for good hypotheses, but also in selecting between actions. Possible actions are treated as disjunctions in the tableau, and the error between the agent’s goals and its simulated expectation is computed, so that the extended tableau algorithm may select the branch with minimal error.

Comirit and the Toy Box Problem

The Comirit Framework combines simulation, logical deduction and machine learning; as such, it is ideally suited to the physical reasoning (well suited to simulation), abstract reasoning (well-suited to tableau-based logical deduction), learning (as parameter search in the tableau) and action selection (as action search in the tableau) in the Toy Box Problem.

There is insufficient space here to provide a detailed analysis of the problem, and indeed, this work is itself ongoing (hence the ‘preliminary’ nature of the solution), however our early results are encouraging.

We conduct an experiment as depicted in Figure 4. A virtual 2D world is simulated with models of simple toys including boxes, balls, bagels and bananas all of varying, weight, appearance and strengths all of which are subject to 2D physics. The agent may only observe the world through 2D raster images (it cannot observe the underlying models), and it must construct its own internal models and predictions. Accuracy is measured by projecting the agent’s belief back into a raster image

and performing a pixel-by-pixel comparison (this is a demanding test since it makes no allowances for ‘nearly’ correct).

In our early experiments we have the system learn two hidden parameters (mass and spring constant). These two parameters are combined in a vector and serve as the hypothesis space for the learning problem. Even though we use an extremely simple machine learning algorithm (stochastic hill-climbing search), a single pair of visual observations is sufficient for the agent to achieve 95% accuracy in predicting an object’s future behavior. This astounding learning rate is depicted in Figure 5. The slight improvement from subsequent observations comes from the elimination of minor overfitting—the accuracy is as good as may be achieved given the differences in the underlying models.

This incredible learning rate is possible because a single pair of images (before and after) contains thousands of pixels, serving as thousands of data-points for training. Indeed, this learning rate aligns with the human competence in develop-

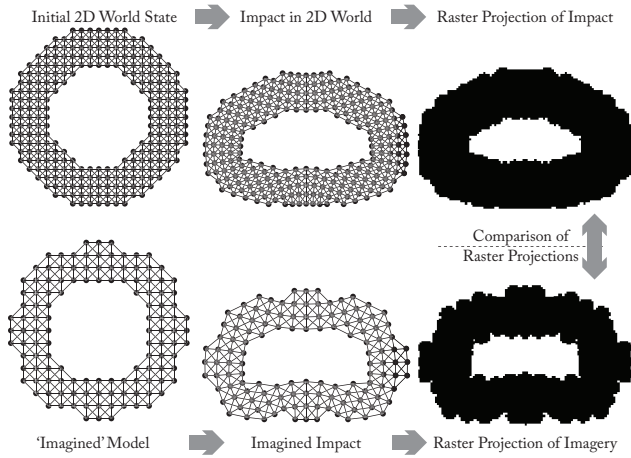


Figure 4: Setup of initial experiments

ing a ‘good intuition’ for an object (whether it is heavy, hard, smooth, fragile, *etc.*) after just a fleeting interaction.

We have also begun exploring the use of naïve Self-Organizing Maps (SOM) (Kohonen 1998) for learning the underlying structure of the world (*e.g.*, that ‘bagels’ are generally light, that balls are often soft, and that metallic objects are usually solid). In this case, the hypothesis becomes an entire SOM, combined with vectors for each visible object. When beliefs about toys are refined, a standard update is applied to the SOM. Our preliminary findings are that the ability for the SOM to generalize across instances roughly doubles the learning rate and provides better initial hypotheses about unknown objects. However, these are early findings and we will report on this in more detail once we have refined the model.

Finally, while the concrete implementation of action selection remains as future work, action selection does not present any theoretical challenge. Given simulation that has been learnt, actions (or sequences of actions) are selected by searching (in an extended tableau) for a sequence that, when performed in simulation, are closest to the agent’s goals (*e.g.*, the goal of tidying-up the toys).

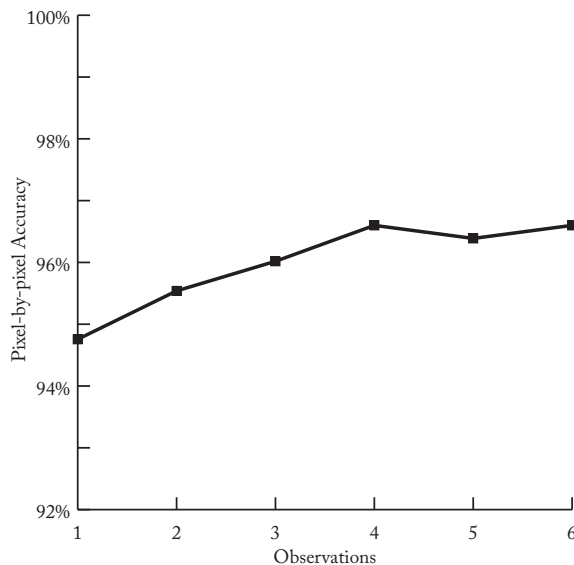


Figure 5: Learning rate in a 2D world

Conclusion

In this paper we have described an open ended benchmark problem that we believe is useful for evaluating and comparing the practical real-world intelligence.

We have also presented a brief overview of the Comirit architecture (with particular emphasis on the recent extensions for learning), and sketched how its capabilities may be applicable to the Toy Box Problem. A comprehensive adaptation and analysis remains, however our early indications (both qualitative and quantitative) suggest that Comirit will be able to ‘solve’ certain instances of the Toy Box Problem. As such, we believe that the pairing E1&T1 are within the realm of plausibility today.

Of course, much future work remains: comprehensive implementation and evaluation, more challenging environments and toys, and the development of methods for learning the fundamental laws of physics and ‘learning to learn’. However, we believe that the Toy Box Problem provides an exciting framework for guiding and evaluating incremental progress towards systems with deep and practical real-world intelligence.

References

- Cassimatis, N. (2005) ‘Integrating Cognitive Models Based on Different Computational Methods’, *Proceedings of the Tenth Annual Conference of the Cognitive Science Society*.
- Gardin, F. and Meltzer, B. (1989) ‘Analogical Representations of Naïve Physics’, *Artificial Intelligence*, vol. 38, pp. 139–59.
- Goertzel, B. and Bugaj, S.V. (2009) ‘AGI Preschool: A Framework for Evaluating Early-Stage Human-like AGIs’, *Proceedings of the Second International Conference on Artificial General Intelligence (AGI-09)*.
- Hähnle, R. (2001) ‘Tableaux and Related Methods’, In Robinson, J.A. and Voronkov, A. (eds.), *Handbook of Automated Reasoning Volume 1*, MIT Press.
- Johnston, B. and Williams, M-A. (2007) ‘A Generic Framework for Approximate Simulation in Commonsense Reasoning Systems’, *Proceedings of the AAAI 2007 Spring Symposium on Logical Formalizations of Commonsense Reasoning (Commonsense 2007)*.
- Johnston, B. and Williams, M-A. (2008) ‘Comirit: Commonsense Reasoning by Integrating Simulation and Logic’, *Proceedings of the First International Conference on Artificial General Intelligence (AGI-08)*.
- Johnston, B. and Williams, M-A. (2009) ‘Autonomous Learning of Commonsense Simulations’, *Proceedings of the Ninth International Symposium on Logical Formalizations of Commonsense Reasoning (Commonsense 2009)*.
- Johnston, B. (2009) *Practical Artificial Commonsense*, PhD Dissertation, Under Examination.
- Kohonen, T. (1998) ‘The Self-Organizing Map’, *Neurocomputing*, no. 21, pp. 1–6.
- Morgenstern, L. and Miller, R. (2009) *The Commonsense Problem Page*, <<http://www-formal.stanford.edu/leora/commonsense/>>, Accessed 13 October 2009.

Grounding Possible Worlds Semantics in Experiential Semantics

Matthew Iklé

Adams State College

Ben Goertzel

Novamente LLC

Abstract

Probabilistic Logic Networks (PLN), a comprehensive framework for uncertain inference currently in use in the OpenCog and Novamente Cognition Engine AGI software architectures, has previously been described in terms of the “experiential semantics” of an intelligent agent embodied in a world. However, several aspects of PLN are more easily interpreted and formulated in terms of “possible worlds semantics”; here we use a formal model of intelligent agents to show how a form of possible worlds semantics can be derived from experiential semantics, and use this to provide new interpretations of several aspects of PLN (including uncertain quantifiers, intensional inheritance, and indefinite probabilities.) These new interpretations have practical as well as conceptual benefits, as they give a unified way of specifying parameters that in the previous interpretations of PLN were viewed as unrelated.

Introduction

The mind of an intelligent agent accumulates knowledge based on experience, yet also creates hypothetical knowledge about “the world as it might be,” which is useful for guiding future actions. This dichotomy – between experience and hypothesis – occurs in regard to many types of knowledge; and in the context of declarative knowledge, it is related to the distinction between experiential and possible-worlds semantics. Here we discuss how these two forms of semantics may be related to each other in the context of a generally intelligent agent that interprets its experience (at least partially) using probabilistic logic. Our treatment pertains specifically to the Probabilistic Logic Networks (PLN) inference framework, which is currently in use in the OpenCog and Novamente Cognition Engine AGI software architectures and has been used for applications including natural language based reasoning (Gea06) and virtual agent reinforcement learning (Goe08); however, many of the points raised could be extended more generally to any probabilistic inference framework.

In much of our prior work on PLN, we have utilized “experiential semantics”, according to which the meaning of each logical statement in an agent’s memory is defined in terms of the agent’s experiences. However

we have also found that certain aspects of PLN are best interpreted in terms of “possible worlds semantics”, in which the meaning of a statement is defined by reference to an ensemble of possible worlds including the one the agent interpreting the statement has experienced. The relation between these two semantic approaches in the PLN context has previously been left informal; the core goal of this paper is to specify it, via providing an *experiential grounding of possible worlds semantics*.

We study an agent whose experience constitutes one “actual world” drawn from an ensemble of possible worlds. We use the idea of bootstrapping from statistics to generate a set of “simulated possible worlds” from the actual world, and prove theorems regarding conditions under which, for a probabilistic predicate F , the truth value of F evaluated over these simulated possible worlds gives a good estimate of the truth value of F evaluated over the ensemble of possible worlds from which the agent’s actual world is drawn.

The reader with a logic background should note that we are construing the notion of possible worlds semantics broadly here, in the philosophical sense (Lew86), rather than narrowly in the sense of Kripke semantics (Gol03) and its relatives. In fact there are interesting mathematical connections between the present formulation and Kripke semantics and epistemic logic, but we will leave these for sequel papers.

Then, we show how this apparatus of simulated possible worlds simplifies the interpretation of several aspects of PLN, providing a common foundation for setting various PLN system parameters that were previously viewed as distinct. We begin with indefinite probabilities (Iea07; Gea08), noting that the second-order distribution involved therein may be interpreted using possible worlds semantics. Then we turn to uncertain quantifiers, showing that the third-order distribution used to interpret these in (IG08) may be considered as a distribution over possible worlds. Finally, we consider intensional inference, suggesting that the complexity measure involved in the definition of PLN intension (Gea08) may be derived from a probability measure over possible worlds. By considering the space of possible worlds implicit in an agent’s experience, one arrives at a simpler unified view of various aspects of the

agent’s uncertain reasoning, than if one grounds these aspects in the agent’s experience directly. This is not an abandonment of experiential semantics but rather an acknowledgement that a simple variety of possible worlds semantics is derivable from experiential semantics, and usefully deployable in the development of uncertain inference systems for general intelligence.

We will not review the PLN inference framework here but will assume the reader has a basic familiarity with PLN terms and notation, as would be found by reading (Gea08) or (Iea07).

A Formal Model of Intelligent Agents

We very briefly review a simple formal model of intelligent agents: Simple Realistic Agent Model (SRAM). Following Legg and Hutter’s framework (LH07), we consider a class of active agents which observe and explore their environment and take actions in it. The agent sends information to the environment by sending symbols from some finite alphabet called the *action space* Σ ; and the environment sends signals to the agent with symbols from an alphabet called the *perception space*, denoted \mathcal{P} . Agents can also experience rewards, which lie in the *reward space*, denoted \mathcal{R} , which for each agent is a subset of the rational unit interval.

To Legg and Hutter’s framework, we add a set \mathcal{M} of memory actions, allowing agents to maintain memories (of finite size), and at each time step to carry out internal actions on their memories as well as external actions in the environment. Further extending the Legg and Hutter framework, we also introduce the notions of *goals* associated with symbols, drawn from the alphabet \mathcal{G} , and *goal-seeking agents*; and we consider the environment as sending goal-symbols to the agent along with regular observation-symbols. We also introduce a conditional distribution $\gamma(g, \mu)$ that gives the weight of a goal g in the context of a particular environment μ .

In this extended framework, an interaction sequence looks like

$$m_1 a_1 o_1 g_1 r_1 m_2 a_2 o_2 g_2 r_2 \dots$$

where the m_i ’s represent memory actions, the a_i ’s represent external actions, the o_i ’s represent observations, the g_i ’s represent agent goals, and the r_i ’s represent rewards. The reward r_i provided to an agent at time i is determined by the goal function g_i . Introducing w as a single symbol denoting the combination of a memory action and an external action, and y as a single symbol denoting the combination of an observation, a goal and a reward, we can simplify this interaction sequence as

$$w_1 y_1 w_2 y_2 \dots$$

Each goal function maps each finite interaction sequence $I_{g,s,t} = wy_{s:t}$ into a value $r_g(I_{g,s,t}) \in [0, 1]$ indicating the value or “raw reward” of achieving the goal during that interaction sequence. The total reward r_t obtained by the agent is the sum of the raw rewards obtained at time t from all goals whose symbols occur in the agent’s history before t .

The agent is represented as a function π which takes the current history as input, and produces an action as output. Agents need not be deterministic and may induce a probability distribution over the space of possible actions, conditioned on the current history. In this case we may characterize the agent by a probability distribution $\pi(w_t | wy_{<t})$. Similarly, the environment may be characterized by a probability distribution $\mu(y_k | wy_{<k})$. The distributions π and μ define a probability measure over the space of interaction sequences.

Following Legg and Hutter, we will consider the class of environments that are *reward-summable*, meaning that the total amount of reward they return to any agent is bounded by 1. We will also use the term “context” to denote the combination of an environment, a goal function and a reward function. If the agent is acting in environment μ , and is provided with $g_t = g$ for the time-interval $T = t \in \{t_1, \dots, t_2\}$, then the *expected goal-achievement* of the agent during the interval is

$$V_{\mu,g,T}^{\pi} \equiv \sum_{t_1}^{t_2} r_i$$

where E is the space of computable, reward-summable environments.

Next, we introduce a second-order probability distribution ν over the space of environments μ . One such distribution is the Solomonoff-Levin universal distribution in which one sets $\nu = 2^{-K(\mu)}$; but this is not the only distribution of interest. A great deal of real-world general intelligence consists of the adaptation of intelligent systems to other particular distributions ν over environment-space (Goe10; Goe09).

Inducing a Distribution over Predicates and Concepts

Given a distribution over environments as defined above, and a collection of predicates evaluated on subsets of environments, we will find it useful to define distributions (induced by the distribution over environments) defining the probabilities of these predicates.

Suppose we have a pair (F, T) where F is a function mapping sequences of perceptions into fuzzy truth values, and T is an integer connoting a length of time. We can define the prior probability of (F, T) as the average degree to which F is true, over a random interval of perceptions of length T drawn from a random environment drawn from the distribution over environments. More generally, if one has a pair (F, f) , where f is a distribution over the integers, one can define the prior probability of (F, f) as the weighted average of the prior probability of (F, T) where T is drawn from f .

While expressed in terms of predicates, the above formulation can also be useful for dealing with concepts, e.g. by interpreting the concept *cat* in terms of the predicate *isCat*. So we can use this formulation in inferences where one needs a concept probability like $P(\text{cat})$ or a relationship probability like $P(\text{eat}(\text{cat}, \text{mouse}))$.

Grounding Possible Worlds Semantics in Experiential Semantics

Now we explain how to ground a form of possible worlds semantics in experiential semantics. We explain how an agent, experiencing a single stream of perceptions, may use this to construct an ensemble of “simulated” possible worlds, which may then be used in various sorts of inferences. This idea is closely related to a commonplace idea in the field of statistics: “subsampling,” a form of “bootstrapping.”

In subsampling, if one has a single dataset D which one wishes to interpret as coming from a larger population of possible datasets, and one wishes to approximately understand the distribution of this larger population, then one generates a set of additional datasets via removing various portions of D . By removing a portion of D , one obtains another dataset. One can then look at the distribution of these auxiliary datasets, considering it as a model of the population D .

This notion ties in closely with SRAM, which considers a probability distribution over a space of environments which are themselves probability distributions. A real agent has a single series of remembered observations. It can induce an approximation of this distribution over environments by subsampling its memory and asking: what would it imply about the world if the items in this subsample were the only things I’d seen?

It may be conceptually useful to observe that a related notion to subsampling is found in the literary methodology of science fiction. Many SF authors have followed the methodology of changing one significant aspect of our everyday world, and depicting the world as they think it might exist if this one aspect were changed (or, a similar methodology may be followed via changing a small number of aspects). This is a way of generating a large variety of alternate possible worlds from the raw material of our own world.

The subsampling and SF analogies suggest two methods of creating a possible world within SRAM (and by repetition, an ensemble of possible worlds) from the agents experience. An agent’s interaction sequence with its environment forms a sample from which it wishes to infer its environment. To better assess this environment, the agent may, for example,

1. create a possible world by removing a randomly selected collection of interactions from the agents memory. In this case, the agent’s interaction sequence would be of the form $I_{g, s, t, (n_t)} = wy(n_t)$ where (n_t) is some subsequence of $1 : t - 1$.
2. create a possible world via assuming a counterfactual hypothesis (i.e. assigning a statement a truth value that contradicts the agents experience), and using inference to construct a set of observations that is as similar to its memory as possible, subject to the constraint of being consistent with the hypothesis.
3. create a possible world by reorganizing portions of the interaction sequence.

4. create a possible world by some combination of the above.

Here we will focus on the first option, leaving the others for future work. We denote an alteration of an iteration sequence $I_{g, s, t}^a$ for an agent a by $\tilde{I}_{g, s, t}^a$, and the set of all such altered interaction sequences for agent a by \mathcal{I}^a .

An agent’s interaction sequence will presumably be some reasonably likely sequence. We would therefore be most interested in those cases for which $d_I(I_{g, s, t}^a, \tilde{I}_{g, s, t}^a)$ is small, where $d_I(\cdot, \cdot)$ is some measure of sequence similarity. The probability distribution ν over environments μ will then tend to give larger probabilities to nearby sequences, than to ones that are far away. An agent would typically be interested in considering only minor hypothetical changes to its interaction sequences, and would have little basis for understanding the consequences of drastic alterations.

Any of the above methods for altering interaction sequences would alter an agent’s perception sequence causing changes to the fuzzy truth values mapped by the function F . This in turn would yield new probability distributions over the space of possible worlds, and thereby yielding altered average probability values for the pair (F, T) . This change, constructed from the perspective of the agent based on its experience, could then cause the agent to reassess its action w . This is what we mean by “experiential possible worlds” or EPW.

The creation of altered interaction sequences may, under appropriate assumptions, provide a basis for creating better estimates for the predicate F than we would otherwise have from a single real-world data point. More specifically we have the following results, which discuss the estimates of F made by either a single agent or a population of agents, based on each agent in the population subsampling their experience.

Theorem 1. *Let \mathcal{E}_n represent an arbitrary ensemble of n agents chosen from \mathcal{A} . Suppose that, on average over the set of agents $a \in \mathcal{E}_n$, the set of values $F(I)$ for mutated interaction sequences I is normal and unbiased, so that,*

$$E[F] = \frac{1}{n} \sum_{a \in \mathcal{E}_n} \sum_{I_{g, s, t}^a \in \mathcal{I}^a} F(I_{g, s, t}^a) P(I_{g, s, t}^a).$$

Suppose further that these agents explore their environments by creating hypothetical worlds via altered interaction sequences. Then an unbiased estimate for $E[F]$ is given by

$$\begin{aligned} \hat{F} &= \frac{1}{n} \sum_{a \in \mathcal{E}_n} \sum_{\tilde{I}_{g, s, t}^a \in \mathcal{I}^a} F(\tilde{I}_{g, s, t}^a) P(\tilde{I}_{g, s, t}^a) \\ &= \frac{1}{n} \sum_{a \in \mathcal{E}_n} \sum_{\tilde{I}_{g, s, t}^a \in \mathcal{I}^a} F(\tilde{I}_{g, s, t}^a) \sum_{e \in E} [P(e|I_{g, s, t}^a) P(\tilde{I}_{g, s, t}^a|e)]. \end{aligned}$$

Proof. That \hat{F} is an unbiased estimate for $E[F]$ follows as a direct application of standard statistical bootstrapping theorems. See, for example, (DE96). \square

Theorem 2. Suppose that in addition to the above assumptions, we assume that the predicate F is Lipschitz continuous as a function of the interaction sequences $I_{g,s,t}^a$. That is,

$$d_F\left(F(\tilde{I}_{g,s,t}^a), F(I_{g,s,t}^a)\right) \leq K d_I(\tilde{I}_{g,s,t}^a, I_{g,s,t}^a),$$

for some bound K and $d_F(\cdot, \cdot)$ is a distance measure in predicate space. Then, setting both the bias correction and acceleration parameters to zero, the bootstrap BC_α confidence interval for the mean of F satisfies

$$\hat{F}_{BC_\alpha}[\alpha] \subset [\hat{F} - Kz^{(\alpha)}\hat{\sigma}_I, \hat{F} + Kz^{(\alpha)}\hat{\sigma}_I]$$

where $\hat{\sigma}_I$ is the standard deviation for the altered interaction sequences and, letting Φ denote the standard normal c.d.f., $z^{(\alpha)} = \Phi^{-1}(\alpha)$.

Proof. Note that the Lipschitz condition gives

$$\begin{aligned} \hat{\sigma}_F^2 &= \frac{1}{n|\mathcal{I}^a| - 1} \times \\ &\sum_{a \in \mathcal{E}_n} \sum_{\tilde{I}_{g,s,t}^a \in \mathcal{I}^a} d_F^2\left(F(\tilde{I}_{g,s,t}^a), F(I_{g,s,t}^a)\right) P(\tilde{I}_{g,s,t}^a) \\ &\leq \frac{K^2}{n|\mathcal{I}^a| - 1} \sum_{a \in \mathcal{E}_n} \sum_{\tilde{I}_{g,s,t}^a \in \mathcal{I}^a} d_I^2(\tilde{I}_{g,s,t}^a, I_{g,s,t}^a) P(\tilde{I}_{g,s,t}^a) \\ &= K^2 \hat{\sigma}_I^2. \end{aligned}$$

Since the population is normal and the bias correction and acceleration parameters are both zero, the BC_α bootstrap confidence interval reduces to the standard confidence interval, and the result then follows (DE96). \square

These two theorems together imply that, on average, through subsampling via altered interaction sequences, agents can obtain unbiased approximations to F ; and, by keeping the deviations from their experienced interaction sequence small, the deviations of their approximations will also be small.

While the two theorems above demonstrate the ability of the subsampling approach to generate probabilistic possible-worlds semantics from experiential semantics, they fall short of being relevant to practical AI inference systems, because the Lipschitz condition in Theorem 2 is an overly strong assumption. With this in mind we offer the following modification, that is more realistic and also in keeping with the flavor of PLN's indefinite probabilities approach. The following theorem basically says that: If one or more agents evaluate the truth value of a probabilistic predicate F via a series of subsampled possible worlds that are normally and unbiasedly distributed around the agent's actual experience, and if the predicate F is mostly smoothly dependent on changes in the world, then evaluating the truth value of F using subsampled possible worlds gives roughly the same results as would be gotten by evaluating the truth value of F across the overall ensemble of possible worlds from which the agent's experience is drawn.

Theorem 3. Define the set

$$I^{a;b} = \left\{ \tilde{I}_{g,s,t}^a \mid d_F^2\left(F(\tilde{I}_{g,s,t}^a), F(I_{g,s,t}^a)\right) = b \right\},$$

and assume that for every real number b the perceptions of the predicate F satisfy

$$\frac{1}{n} \sum_{a \in \mathcal{E}_n} P(I^{a;b}) \leq \frac{M(b)}{b^2} \sigma_I^2$$

for some $M(b) \in \mathbb{R}$. Further suppose that

$$\int_0^1 M(b) db = M^2 \in \mathbb{R}.$$

Then under the same assumptions as in Theorem 1, and again setting both the bias correction and acceleration parameters to zero, we have

$$\hat{F}_{BC_\alpha}[\alpha] \subset [\hat{F} - M\sqrt{n}z^{(\alpha)}\hat{\sigma}_I, \hat{F} + M\sqrt{n}z^{(\alpha)}\hat{\sigma}_I]$$

Proof.

$$\begin{aligned} \hat{\sigma}_F^2 &= \frac{1}{n \cdot |\mathcal{I}^a| - 1} \times \\ &\sum_{a \in \mathcal{E}_n} \sum_{\tilde{I}_{g,s,t}^a \in \mathcal{I}^a} d_F^2\left(F(\tilde{I}_{g,s,t}^a), F(I_{g,s,t}^a)\right) P(\tilde{I}_{g,s,t}^a) \\ &= \frac{1}{n \cdot |\mathcal{I}^a| - 1} \times \\ &\sum_{a \in \mathcal{E}_n} \int_0^1 \sum_{\tilde{I}_{g,s,t}^a \in \mathcal{I}^{a;b}} d_F^2\left(F(\tilde{I}_{g,s,t}^a), F(I_{g,s,t}^a)\right) P(\tilde{I}_{g,s,t}^a) db \\ &\leq \frac{1}{n \cdot |\mathcal{I}^a| - 1} \times \\ &\sum_{a \in \mathcal{E}_n} \int_0^1 \sum_{\tilde{I}_{g,s,t}^a \in \mathcal{I}^{a;b}} d_F^2\left(F(\tilde{I}_{g,s,t}^a), F(I_{g,s,t}^a)\right) P(\tilde{I}_{g,s,t}^a) db \\ &\leq b^2 n \frac{M^2}{b^2} \sigma_I^2 = (M\sqrt{n})^2 \sigma_I^2. \end{aligned}$$

\square

In the following sections we show how this new formalization of possible worlds semantics can be used to clarify the conceptual and mathematical foundations of several aspects of PLN inference.

Reinterpreting Indefinite Probabilities

Indefinite probabilities (Iea07; Gea08) provide a natural fit with the experiential semantics of the SRAM model, as well as with the subsampling methodology articulated above. An indefinite probability truth-value takes the form of a quadruple $([L, U], b, k)$. The meaning of such a truth-value, attached to a statement S is, roughly: There is a probability b that, after k more observations, the truth value assigned to the statement S will lie in the interval $[L, U]$. We interpret an interval $[L, U]$ by assuming some particular family of distributions (usually Beta) whose means lie in $[L, U]$.

To execute inferences using indefinite probabilities, we make heuristic distributional assumptions, assuming a “first order distribution of means, with $[L, U]$ as a (100b)% credible interval. Corresponding to each mean in this “first-order” distribution is a “second order distribution, providing for an “envelope” of distributions.

The resulting bivariate distribution can be viewed as an heuristic approximation intended to estimate unknown probability values existing in hypothetical future situations. Combined with additional parameters, each indefinite truth-value object essentially provides a compact representation of a single second-order probability distribution with a particular, complex structure.

In the EPW context, the second-order distribution in an indefinite probability is most naturally viewed as a distribution over possible worlds; whereas, each first-order distribution represents the distribution of values of the proposition within a given possible world.

As a specific example, consider the case of two virtual agents: one agent, with cat-like characteristics, called “Fluffy” and the second a creature, with dog-like characteristics, named “Muffin.” Upon a meeting of the two agents, Fluffy might immediately consider three courses of action: Fluffy might decide to flee as quickly as possible, might hiss and threaten Muffin, or might decide to remain still. Fluffy might have a memory store of perception sequences from prior encounters with agents with similar characteristics to those of Muffin.

In this scenario, one can view the second-order distribution as a distribution over all three courses of action that Fluffy might take. Each first-order distribution would represent the probability distribution of the result from the corresponding action. By hypothetically considering all three possible courses of action and the probability distributions of the resulting action, Fluffy can make more rational decisions.

Reinterpreting Indefinite Quantifiers

EPW also allows PLN’s universal, existential and fuzzy quantifiers to be expressed in terms of implications on fuzzy sets. For example, if we have

ForAll $\$X$

Implication

Evaluation $F \$X$

Evaluation $G \$X$

then this is equivalent to

AverageQuantifier $\$X$

Implication

Evaluation $F^* \$X$

Evaluation $G^* \$X$

where e.g. F^* is the fuzzy set of variations on F constructed by assuming possible errors in the historical evaluations of F . This formulation yields equivalent results to the one given in (Gea08), but also has the property of reducing quantifiers to FOPLN (over sets derived from special predicates).

To fully understand the equivalence of the above two expressions, first recall that in (Gea08), we handle quantifiers by introducing third-order probabilities. As discussed there, the three levels of distributions are roughly as follows. The first- and second-order levels play the role, with some modifications, of standard indefinite probabilities. The third-order distribution then plays the role of “perturbing the second-order distribution. The idea is that the second-order distribution represents the mean for the statement $F(x)$. The third-order distribution then gives various values for x , and the first-order distribution gives the sub-distributions for each of the second-order distributions. The final result is then found via an averaging process on all those second-order distributions that are “almost entirely” contained in some *ForAll_proxy_interval*.

Next, *AverageQuantifier* $F(\$X)$ is a weighted average of $F(\$X)$ over all relevant inputs $\$X$; and we define the fuzzy set F^* as the set of perturbations of a second-order distribution of hypotheses, and G^* as the corresponding set of perturbed implication results. With these definitions, not only does the above equivalence follow naturally, so do the “possible/perturbed worlds” semantics for the ForAll quantifier. Other quantifiers, including fuzzy quantifiers, can be similarly recast.

Specifying Complexity for Intensional Inference

A classical dichotomy in logic involves the distinction between extensional inference (involving sets with members) and intensional inference (involving entities with properties). In PLN this is handled by taking extension as the foundation (where, in accordance with experiential semantics, sets ultimately boil down to sets of elementary observations), and defining intension in terms of certain fuzzy sets involving observation-sets. This means that in PLN intension, like higher-order inference, ultimately emerges as a subcase of FOPLN (though a subcase with special mathematical properties and special interest for cognitive science and AI). The prior formulation of PLN intension contains a “free parameter” (a complexity measure) which is conceptually inelegant; EPW remedies this via providing this parameter with a foundation in possible worlds semantics.

To illustrate how, in PLN, higher-order intensional inference reduces to first-order inferences, consider the case of intensional inheritance. *IntensionalInheritance* $A B$ measures the extensional inheritance between the set of properties or patterns associated with A and the corresponding set associated with B . This concept is made precise via formally defining the concept of “pattern,” founded on the concept of “association.” We formally define the association operator ASSOC through:

ExtensionalEquivalence

Member $\$E$ (ExOut ASSOC $\$C$)

ExOut

Func

List
 Inheritance \$E \$C
 Inheritance
 NOT \$E
 \$C

where $\text{Func}(x, y) = [x - y]^+$ and $+$ denotes the positive part.

We next define a pattern in an entity A as something that is associated with, but simpler than, A. Note that this definition presumes some measure $c()$ of complexity. One can then define the fuzzy-set membership function called the “pattern-intensity,” via

$$IN(F, G) = [c(G) - c(F)]^+ [P(F|G) - P(F| \neg G)]^+.$$

The complexity measure c has been left unspecified in prior explications of PLN, but in the present context we may take it as the measure over concepts implied by the measure over possible worlds derived via subsampling as described above (or perhaps by counterfactuals).

Reinterpreting Implication between Inheritance Relationships

Finally, one more place where possible worlds semantics plays a role in PLN is with implications such as

Implication

Inheritance Ben American
 Inheritance Ben obnoxious

We can interpret these by introducing predicates over possible worlds, so that e.g.

$$Z_{\text{Inheritance Ben American}}(W) < t >$$

denotes that t is the truth value of *Inheritance Ben American* in world W . A prerequisite for this is that *Ben* and *American* be defined in a way that spans the space of possible worlds in question. When defining possible worlds by differing subsets of the same observation-set, this is straightforward; in the case of possible worlds defined via counterfactuals it is subtler and we omit details here.

The above implication may then be interpreted as

AverageQuantifier \$W

Implication

Evaluation $Z_{\text{Inheritance Ben obnoxious}} \W
 Evaluation $Z_{\text{Inheritance Ben American}} \W

The weighting over possible worlds $\$W$ may be taken as the one obtained by the system through the subsampling or counterfactual methods as indicated above.

Conclusion

We began with the simple observation that the mind of an intelligent agent accumulates knowledge based on experience, yet also creates hypothetical knowledge about “the world as it might be,” which is useful for guiding

future actions. PLN handles this dichotomy via a foundation in experiential semantics, and it is possible to formulate all PLN inference rules and truth value formulas in this way. Some PLN truth value formulas are simplified by interpreting them using possible world semantics. With this in mind we used subsampling to define a form of experientially-grounded possible-worlds semantics, and showed its use for handling indefinite truth values, probabilistic quantifiers and intensional inference. These particular technical ideas illustrate the more general thesis that a combination of experiential and possible-worlds notions may be the best approach to comprehending the semantics of declarative knowledge in generally intelligent agents.

References

- Thomas J. DiCiccio and Bradley Efron. Bootstrap confidence intervals. *Statistical Science*, Vol. 11, No. 3, 189-228, 1996.
- Ben Goertzel and Hugo Pinto et al. Using dependency parsing and probabilistic inference to extract gene/protein interactions implicit in the combination of multiple biomedical research abstracts. In *Proceedings of BioNLP-2006 Workshop at ACL-2006, New York*, 2006.
- Ben Goertzel and Matthew Iklé et al. *Probabilistic Logic Networks*. Springer, 2008.
- Ben Goertzel. An integrative methodology for teaching embodied non-linguistic agents, applied to virtual animals in second life. In *Proceedings of the First AGI Conference, Memphis*, 2008.
- Ben Goertzel. The embodied communication prior. *Proceedings of ICCI 2009, Hong Kong*, 2009.
- Ben Goertzel. Toward a formalization of real-world general intelligence. *Submitted to AGI-10*, 2010.
- Robert Goldblatt. Mathematical modal logic: a view of its evolution. *Journal of Applied Logic 1: 30992*, 2003.
- Matthew Iklé and Ben Goertzel et al. Indefinite probabilities for general intelligence. In *Proceedings of 2006 AGIRI conference, Bethesda, MD: IOS Press*, 2007.
- Matthew Iklé and Ben Goertzel. Probabilistic quantifier logic for general intelligence: An indefinite probabilities approach. In *Proceedings of AGI-08. Memphis*, 2008.
- David Lewis. *On the Plurality of Worlds*. Basil Blackwell, 1986.
- Shane Legg and Marcus Hutter. A formal measure of machine intelligence. In *Proceedings of Benelam-2006, Ghent*, 2007.

Relational Local Iterative Compression

Laurent Orseau

UMR AgroParisTech / INRA 518

AgroParisTech

16 rue Claude Bernard, Paris, France

Compression in the program space is of high importance in Artificial General Intelligence [Sol64, Hut07]. Since maximal data compression in the general sense is not possible to achieve [Sol64], it is necessary to use approximate algorithms, like $AIXI_{t,l}$ [Hut07].

This paper introduces a system that is able to compress data locally and iteratively, in a relational description language. The system thus belongs to the anytime algorithm family: the more time spent, the better it performs. The locality property is also well-suited for AGI agents to allow them to focus on "interesting" parts of the data.

The system presented here is to be opposed to blind generate and test approaches (e.g., [Sch04, Lev73]). On the contrary to the latter, it uses information gathered about the input data to guide compression. It can be described as a forward chaining¹ expert system on relational descriptions of input data, while looking for the most compressed representation of the data.

It is composed of a description/programming language, to describe facts (and a set of weights associated with each primitive of the language), local search operators, to infer new facts, and an algorithm to search for compressed global description.

The relation operators and the search operators are domain-specific. Examples in the letter-string domain are given in the *Experiments* section. Due to lack of space, only a overview of the whole system can be given.

Description Language

The main point of this paper is to deal with *local compression*. This means that the system should be able to focus on any part of the input data, without affecting the rest of the data.

A relational language for data representation/programming is well suited for this purpose, exactly because everything (including spatial and dynamical dependencies) can be described in terms of local relations between data parts.

The language has values (numbers, characters, operators names, ...) and relations between objects (instan-

tiations of operators on objects). What kinds of objects and operators are used depends on the domain. For an AGI, it depends on the sensors it uses, but the set of operators should form a Turing-complete language.

The *initial description* of the *world* (the input data) is the initial facts of the expert system.

Search Operators

The inference rules of the expert systems are called the *search operators*. A search operator takes inputs, tests them against a precondition, and when the test is passed produces outputs that are added to the fact database. The set of search operators is domain-dependent.

The exact inputs/outputs mapping is also memorized to construct a graph for the compression part of the algorithm.

The constraint imposed on search operators is that they must not lose information, i.e. that knowledge of the outputs is sufficient to reconstruct the inputs.

Algorithm

The algorithm runs like this:

1. The input data is given to the system in a simple uncompressed relational representation.
2. Each local search operator is tested in turn to create new local descriptions when possible.
3. Local descriptions are composed to create global descriptions.
4. The description (space) costs of the global descriptions are computed.
5. The less costly global description is retained.
6. Stop when a given criterion (on time, space, error, ...) is satisfied, or go back to step 2.

Finding the best current description is based on the Minimum Description Length principle [GMP05], where the cost of a description is simply the sum of the costs of each relation used. The cost of a relation is domain specific, and defined by the user.

¹There can be no backward chaining, because no goal description is given.

Lossless Compression Sketch Proof

Search operators consume inputs and provide outputs (new local descriptions). If each such operator has the property that it does not lose information, i.e. that its inputs can be rebuilt from the outputs, then the global algorithm ensures that no information is lost for global descriptions. The only difficulty resides in cyclic dependencies between local descriptions, e.g. when a local description A depends on the local description B and vice-versa. To avoid such cycles, a dependency directed graph of input-output mappings created by the search operators is constructed, and any cycle is broken. The final description is composed of the relations that are on the terminal nodes of the graph. So some inputs that should have been consumed can appear in the final description because they are needed to avoid a cycle.

Experiments

The system has been tested in the letter-string domain on a set of small strings that show that compression is indeed iterative.

In the letter-string domain, the world is initially described using the following relations:

obj: binds values that describe one same "object", character, of the world (the string),

val: character value a, b, c ...

pos: position of the object in the string,

Once compression has begun, some of the following relations may be used:

neighbor: two objects are neighbors,

succ, **pred**: two values follow one another in lexicographical order

eq: two values are identical,

plus relations to describe sequences (with a initial value, a length and a succession relation) and sequences of sequences.

The letter-string domain search operators have similar names to the description relations. For example, the **eq** search operator searches the fact database for identical values. When it finds one, it creates a new fact using the **eq** relation binding the two values and adds it to the database. The **seqV** search operator searches for two objects that are neighbors and have a relation on their values and creates a new sequence of two objects, whereas **seqG** tries to merge two existing neighbor sequences that have identical succession operators.

For example, the initial string **abcxxxdefyyyy** has a cost of 28 (given a fixed set of costs for the letter-string domain). After one pass on the loop of the algorithm, the system compresses it to a cost of 24.9, finding local relations like **neighbor** and **eq**. On the next loop step, it finds other relations like small sequences but they do not build a less costly description. On the next steps, the sequences grow, lowering the best description cost to 18.8, then 17.6 and finally 14.3, where the string has been "understood"

as **(abc)(xxxx)(def)(yyyy)** with **succ** relations between interleaving sequences and **neighbor** relations between adjacent sequences.

The system also compresses non-obvious strings like **abccdedefg**, on which it lowers the initial cost of 20 to 8.3 with 7 intermediate values, finally finding the compressed representation of the sequence of sequences **((a)(bc)(cde)(defg))**.

Limitations, Perspectives and Conclusion

For the experiments in the letter-string domain, a few seconds are sufficient to find a much compressed description, but lengthening the initial strings leads to a huge combinatorial explosion. To limit the impact of such explosion, the first solution is to add ad-hoc domain-specific search operators that focus on specific "interesting" patterns in the database and are given high priority. It is also possible to add a learning strategy, for example inspired by Explanation Based Learning [DM86], since compressing is equivalent to proving. Learning would lead, with an AGI approach, to Incremental Learning (e.g. [Sch04]), using acquired knowledge to solve related problems faster. Learning could then also be used to incrementally tune the initial costs of the relation operators like **eq**.

The language used for the experiments can represent complex non-linear worlds, but the language should be augmented to Turing-completeness since for an AGI this seems to be unavoidable.

Relational local iterative compression is a novel approach to compression in the program space and could be used for many different tasks, e.g. visual scene (2D, 3D) compression/comprehension, amongst others. It may be mostly beneficial when prior domain knowledge can be used or acquired.

References

- [DM86] G. Dejong and R. Mooney. Explanation-based learning: An alternative view. *Machine Learning*, 1(2):145–176, 1986.
- [GMP05] P. D. Grünwald, I. J. Myung, and M. A. Pitt. *Advances in Minimum Description Length: Theory and Applications*. MIT Press, 2005.
- [Hut07] M. Hutter. Universal algorithmic intelligence: A mathematical top-down approach. In *Artificial General Intelligence*, pages 227–290. Springer Berlin Heidelberg, 2007.
- [Lev73] L. A. Levin. Universal sequential search problems. *Problems of Information Transmission*, 9(3):265–266, 1973.
- [Sch04] J. Schmidhuber. Optimal ordered problem solver. *Machine Learning*, 54(3):211–254, 2004.
- [Sol64] R. J. Solomonoff. A formal theory of inductive inference. *Information and Control*, 7(1):1–22, 1964.

Searching for Minimal Neural Networks in Fourier Space

Jan Koutník, Faustino Gomez, and Jürgen Schmidhuber

IDSIA, Galleria 2, 6928 Manno-Lugano, Switzerland
University of Lugano & SUPSI, Switzerland

Abstract

The principle of minimum description length suggests looking for the *simplest* network that works well on the training examples, where simplicity is measured by network description size based on a reasonable programming language for encoding networks. Previous work used an assembler-like universal network encoding language (NEL) and Speed Prior-based search (related to Levin’s Universal Search) to quickly find low-complexity nets with excellent generalization performance. Here we define a more natural and often more practical NEL whose instructions are frequency domain coefficients. Frequency coefficients may get encoded by few bits, hence huge weight matrices may just be low-complexity superpositions of patterns computed by programs with few elementary instructions. On various benchmarks this weight matrix encoding greatly accelerates the search. The scheme was tested on pole-balancing, long-term dependency T-maze, and ball throwing. Some of the solutions turn out to be unexpectedly simple as they are computable by fairly short bit strings.

Introduction

Given some training experience, what is the best way of computing a weight matrix for a neural network such that it will perform well on unseen test data? Let us ignore for the moment the numerous gradient and evolution based training methods (both with or without teachers), and focus on the essential. The principle of minimum description length (MDL) [WB68, Ris78, LV97] suggests one should search for the *simplest* network that works well on the training examples, where simplicity is measured by the description size of the network, in a reasonable (possibly universal) programming language. In theory, the simplest or most compressible weight matrix for a given problem is the one with lowest algorithmic information or Kolmogorov complexity, i.e. the one computable by the shortest program. Unfortunately, there is no general way of finding this program, due to lack of an upper bound on its runtime [Sol64, Kol65, LV97].

However, there is a theoretically “best” way of taking runtime into account [Lev73, Sch02]. This is the basis of previous work on optimal search for simple networks [Sch95, Sch97], which used an assembler-like universal network encoding language (NEL) and Speed Prior-based search [Sch02] (related to Levin’s Universal Search

[Lev73]), to quickly find low-complexity weight matrices with excellent generalization performance.

In related work, in the context of neuroevolution [Gru92, GS07, BKS09], less general NELs have been used to encode network parameters indirectly in symbol strings which are evolved using a genetic algorithm. Like the early work [Sch95, Sch97], these approaches allow short descriptions to specify networks of arbitrary size.

Here we define a NEL whose instructions are bit representations of Fourier series coefficients, and network weight matrices are computed by applying inverse Fourier-type transforms to the coefficients. This not only yields continuity (a small change to any coefficient changes all weights by a small amount) but also allows the algorithmic complexity of the weight matrix to be controlled by the number of coefficients. As frequency domain representations decorrelate the signal (weight matrix), the search space dimensionality can be reduced in a principled manner by discarding high-frequency coefficients, as is common lossy image coding (note that ignoring high frequencies in the initial phase is encouraged by observations of factorial redundancy in trained weight matrices [Rad93]). Therefore, the search for a good weight matrix can be performed systematically starting with smooth weight matrices containing only low frequencies, and then successively adding higher frequencies.

Encoding in the frequency domain also means that the size of the program is independent of the size of the network it generates, so that networks can be scaled to high-dimensional problems, such as vision, since a very short program consisting of frequency coefficients, each encoded by a few bits, can compute huge weight matrices. While this is the main motivation for most indirect network encoding schemes, here we consider indirect encoding in the opposite direction: given a problem for which a relatively small network solution is known, is there a short encoding that allows the network space to be searched exhaustively?

The next section describes the neural network encoding scheme in detail and the variant of universal search used to find solutions. We then present experimental results in three test domains, showing how some of the solutions turn out to be surprisingly simple, as they are computable by fairly short, network-computing bit strings.

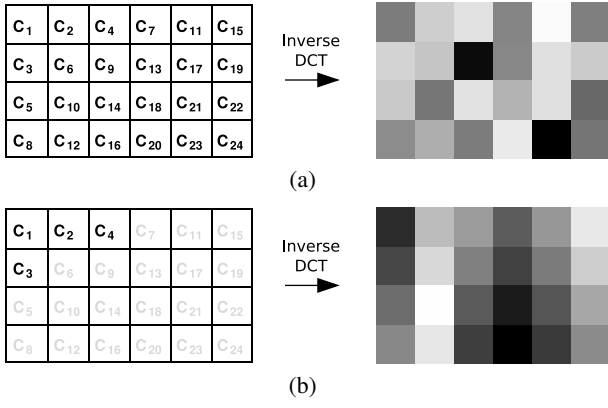


Figure 1: **DCT network representation.** The coefficients are selected according to their order along the second diagonals, going from upper-left corner to the bottom right corner. Each diagonal is filled from the edges to the center starting on the side that corresponds to the longer dimension. (a) Shows an example of the kind of weight matrix (right) that is obtained by transforming the full set of coefficients (left). The grayscale levels denote the weight values (black = low, white = high). (b) Shows the weight matrix when only the first four coefficients from (a) are used. The weights in (b) are more spatially correlated than those in (a).

Searching in Compressed Network Space

The motivation for representing weight matrices as frequency coefficients is that by spatially decorrelating the weights in the frequency domain it might be possible to discard the least significant frequencies, and thereby reduce the number of search dimensions. This in turn makes it possible to search “universally” from small networks that can be represented by few coefficients, to larger networks requiring more complex weight matrices.

The next two sections describe how the networks are represented in the frequency domain using the Discrete Cosine Transform (DCT), and the version of universal search that is used to systematically find solutions to the experiments that follow.

DCT Network Representation

All networks are fully connected recurrent neural networks (FRNNs) with i inputs and single layer of n neurons where some of the neurons are treated as output neurons. This architecture is general enough to represent e.g. feed-forward and Jordan/Elman networks, as they are just sub-graphs of the FRNN.

An FRNN consists of three weight matrices: an $n \times i$ input matrix, \mathbf{I} , an $n \times n$ recurrent matrix, \mathbf{R} , and a bias vector \mathbf{t} of length n . These three matrices are combined into one $n \times (n + i + 1)$ matrix, and encoded indirectly using $c \leq N$ DCT coefficients, where N is the total number of weights in the network. Figure 1 illustrates the relationship between the coefficients and weights for a hypothetical 4×6 weight matrix. The left side of the figure shows two

Algorithm 1: Universal Network Search (r)

```

1 for  $x \leftarrow 1$  to  $2^{2^r}$  do
2   for  $n \leftarrow n_{min}$  to  $x$  do
3     for  $b \leftarrow 1$  to  $x$  do
4       for  $c \leftarrow 1$  to  $\text{MIN}(b, N)$  do
5         if  $\text{MAX}(n, b, c) = x$  then
6           for  $s \leftarrow 1$  to  $2^b$  do
7              $D \leftarrow \text{DECODE}(n, c, \text{BINARY}(s))$ 
8              $network \leftarrow \text{INVERSE DCT}(D)$ 
9             if  $\text{SOLVED?}(\text{EVALUATE}(network))$  then
10              return  $\text{ENCODE}(r, n, b, \text{BINARY}(s))$ 
11          end

```

weight matrix encodings that use different numbers of coefficients $\{C_1, C_2, \dots, C_c\}$. Generally speaking, coefficient C_i is considered to be more significant (associated with a lower frequency) than C_j , if $i < j$. The right side of the figure shows the weight matrices that are generated by applying the inverse DCT transform to the coefficients. In the first case (figure 1a), all of the 24 coefficients is used, so that any possible 4×6 weight matrix can be represented. The particular weight matrix shown was generated from random coefficients in $[-20, 20]$. In the second case (figure 1b), each C_i has the same value as in figure 1a, but the full set has been truncated to only the four most significant coefficients.

The more coefficients, the more high frequency information that is potentially expressed in the weight matrix, so that the weight values become less spatially correlated—large changes can occur from one weight to its neighbors. As c approaches one, the matrix becomes more regular, with only gradual, correlated, changes in value from weight to weight.

Universal Network Search

In order to search the space of networks universally, a strict total ordering must be imposed on the possible DCT encodings. We accomplish this by representing the c coefficients using a total of b bits, and iterating over all possible bit-strings using Universal Network Search (UNS), described in Algorithm 1. The outer-most loop imposes an upper limit, x , for n , b and c . The next three loops examine all combinations of neurons, bits and coefficients, constrained by, n_{min} , the number of output units required by problem in question (second loop), and, N , the total number of weights in the network. Each of the 2^b bit-strings (third loop) is partitioned in b different ways (fourth loop); each partitioning denoting a different number of coefficients. If $(b \bmod c) \neq 0$ then the modulo is distributed into the coefficients from the beginning. For example, if $b = 3$, each of the $2^3 = 8$ possible bit-strings has three possible partitionings: (1) only one coefficient, C_1 , is represented using all three bits, (2) two coefficients, C_1 using two bits, and C_2 using the remaining bit, and (3) three coefficients, C_1, C_2 and C_3 , each using one bit. The set of values that a coefficient C can take on is determined by dividing $[-\alpha, \alpha] \in \mathbb{R}$ into $1/(2^{\ell(C)} - 1)$ intervals, where $\ell(C)$ is the number of bits used to represent C , and α is just a scaling factor. For example, if $\ell(C) = 2$ and

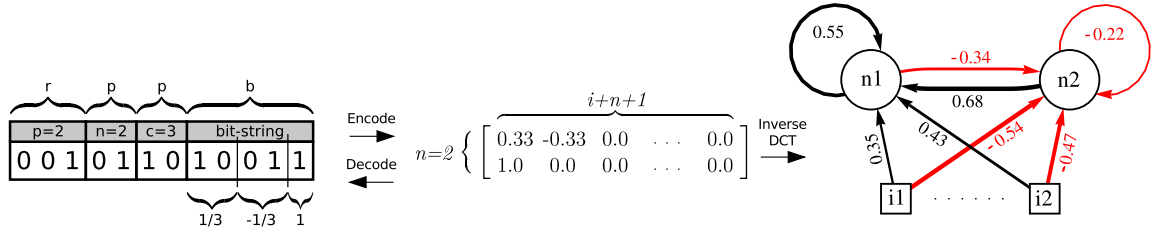


Figure 2: **Network representation and encoding.** A network with two neurons (right) is obtained by applying the inverse DCT of the matrix (middle), where three of the coefficients (C_1, C_2, C_3) are non-zero, in this example. The weight matrix can be encoded (left) by a total of 12 bits, five for the coefficient values, and seven for the “meta” information: three bits for the precision, p , which determines the size of the bit-fields representing n , the number of neurons, and c , the number of coefficients. This is all of the information needed to reconstruct (decode) the complete network.

$\alpha = 1.0$, then the set of values C can take is $\{-1, -\frac{1}{3}, \frac{1}{3}, 1\}$.

Finally, in the inner-most loop, each of the 2^b networks specified by each unique (n, b, c) is decoded into a coefficient matrix D , which is then transformed into a weight matrix via the inverse DCT. The search terminates if either $x > 2^{2^r}$ or a network that solves the problem is found, in which case the successful network is encoded as described in figure 2, and returned.

To completely describe a network, simply storing the bit-string b is not sufficient, the number of neurons, n , and coefficients, c , must be encoded as well. To encode this information in minimal way, we first encode the number of bits that will be used to represent the parameters and then store the parameters with the fixed number of bits. The bit-string that completely describes the network consists of the following fields (see figure 2): r bits represent the bit precision, p , of the n and c fields, p bits each for n and c , and b bits for the actual coefficient values, $b \geq c$, for a total of $r + 2p + b$ bits, where $p \leq 2^r$. For the example 001 01 10 10011, shown in figure 2, the first field has size $r = 3$, and a decimal value of 2, so that n and c are represented by 2 bits, with values of 2 and 3, respectively, meaning that the network has 2 neurons, where 3 coefficients are described with by the last five bits 10011.

A universal search over all possible bit-strings would needlessly examine a large number of invalid bit-strings (having $b < c$). Therefore, we use Algorithm 1 which constrains the search to only valid, decodable strings, and is therefore an instance of Practical Universal Search [SS10].

Experimental Results

Universal Network Search was tested on three tasks: Markovian and non-Markovian pole balancing, the long term dependency T-maze, and the ball throwing task. In all experiments, the scaling factor, α , was set to 20, and the number of bits, r , used to represent the precision of n and c was set to three, which means that the search can continue up to networks with $2^{2^3} = 256$ neurons, more than enough for the tasks in question. In each task, the encoding scheme described in the previous section is used to quantify the complexity of network solutions, and is indicated by the “Total Bits” column in the tables.

Table 1: **Pole balancing results.** Each row describes the minimal (1-neuron) network solution for each task, the number of evaluations that UNS required to find it, and the total number of bits required to encode it. Notice that just 8 evaluations are needed to find a solution to the single pole Markov task.

Task	b	c	Eval.	Total Bits
1 pole Markov	2	2	8	7
1 pole non-Markov	6	3	290	13
2 poles Markov	16	6	773,070	25
2 poles non-Markov	17	5	1,229,012	26

Pole Balancing

Pole balancing (figure 3a) is a standard benchmark for learning systems. The basic version consists of a single pole hinged to a cart, to which a force must be applied in order to balance the pole while keeping the cart within the boundaries of a finite stretch of track. By adding a second pole next to the first, that task becomes much more non-linear and challenging. A further extension is to limit the controller to only have access to the position of the cart, and the angle of the pole(s), and not the velocity information, making the problem non-Markovian (see [Wie91] for setup and equations of motion). The task is considered solved if the pole(s) can be balanced for 100,000 time steps.

Table 1 summarizes the results for the four most commonly used versions of the task. For the Markov single pole task, a successful network with just one neuron whose weights are represented by 2 DCT coefficients is found after just 8 evaluations. This result shows how simple the single pole balancing is: the single neuron, which solves it has monotonically distributed weights. Non-Markovian single pole balancing increases complexity of the task only slightly.

For the two-pole versions, 16 (17 for non-Markovian case) bits are required to solve the problem using a single neuron. Notice that the solution to the Markovian 2-pole task, requiring 8 weights (6 input + 1 recurrent + 1 threshold), has been compressed to 6 parameters, C_1, \dots, C_6 . The non-Markovian 2-pole network has 5 weights and 5 coefficients were used, meaning that in this task it does not DCT

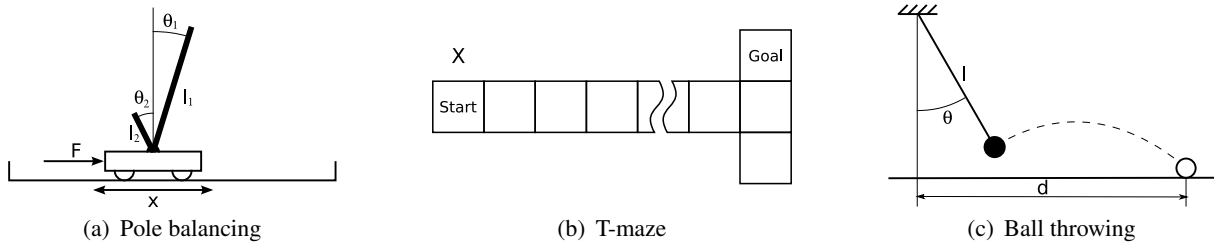


Figure 3: **Evaluation tasks.** (a) Pole balancing: the goal is to apply a force F to the cart such that the pole(s) do not fall down. (b) T-maze: the agent must travel down the corridor remembering the signal X which indicates the location of the goal. The length of the corridor is variable. (c) Ball throwing: the ball attached to the end of the arm must be thrown as far as possible by applying a torque to the joint and then releasing the ball.

Table 2: **T-maze results.** The table shows the two networks with the shortest bit descriptions, found by UNS. The check marks in the “T-maze length” column indicate the corridor lengths the network was able to solve. Note that the seven neuron network is found before the four neuron network since it requires fewer bits to encode (19 vs. 21).

n	b	c	Eval.	T-maze length			Total Bits
				5	50	1000	
7	10	10	85,838	✓	✓	✓	19
4	12	11	306,352	✓	–	–	21

compress the weight matrix. In the other words, the number of bits per coefficient is the restriction which makes the exhaustive search possible.

Long Term Dependency T-maze

The T-maze task is a discrete non-Markovian problem consisting of a corridor of n rooms with a start state S at one end and a T-junction at the opposite end (figure 3b). Starting in S , the objective is to travel down to the end of the corridor and go either north or south at the T-junction depending on a signal X received in S indicating the location of the goal G . In order to choose the correct direction at the junction, the network must remember X for at least n time-steps.

The agent always sees a binary vector of length three. At the start, the observation is either 011 if the goal is to the north, or 110 if it is to the south. In the corridor, the agent sees 101 and at the junction it sees 010. The agent RNN has three output units, one for each of the possible actions (go east, north or south), where the action corresponding to the unit with the highest activation is taken at each time-step. The agent receives a reward of -0.1 if tries to go north or south in the corridor, or go east or in wrong direction at the T-junction, and a reward of 4.0 if it achieves the goal.

All agents are initially evaluated on a corridor of length 5. If the agents achieve the goal, they are also evaluated in corridors of length 50 and 1000 in order to test the generalization ability.

Table 2 shows the results. A four neuron RNN described with 21 bits can achieve the goal in a corridor of length of

Table 3: **Ball throwing results.** The table shows the first three network near optimal networks found by UNS (all network have two neurons). The d column indicates the distance, in meters, the ball was thrown by each network using the strategy indicated in the first column (compare to the distance, d_{opt} , of the corresponding optimal controller).

strategy	b	c	Eval.	d [m]	d_{opt} [m]	Total Bits
fwd	4	4	70	4.075	5.391	11
bwd-fwd	8	8	2516	5.568	5.391	17
bwd-fwd	9	9	5804	9.302	10.202	20

5. A network with seven neurons described with 19 bits can find the goal in a maze of any length (the outputs of the network become stable while in the corridor and the input pattern at the end causes a recall of the goal position stored in the network activation). The 7-neuron network was found before the 4-neuron network because it requires fewer bits and coefficients.

Ball Throwing

In the ball throwing task (figure 3c), the goal is to swing a one-joint artificial arm by applying a torque to the joint, and then releasing the ball such that it is thrown as far as possible. The arm-ball dynamical system is described by:

$$(\dot{\theta}, \dot{\omega}) = \left(\omega, \underbrace{-c \cdot \omega}_{\text{friction}} - \underbrace{\frac{g \cdot \sin(\theta)}{l}}_{\text{gravity}} + \underbrace{\frac{T}{m \cdot l^2}}_{\text{torque}} \right)$$

where θ is the arm angle, ω its angular speed, $c = 2.5\text{s}^{-1}$ the friction constant, $l = 2\text{m}$ the arm length, $g = 9.81\text{ms}^{-2}$, $m = 0.1\text{kg}$ the mass of the ball, and T the torque applied ($T_{\text{max}} = [-5\text{Nm}, 5\text{Nm}]$). In the initial state, the arm hangs straight down ($\theta = 0$) with the ball attached to the end. The controller sees (θ, ω) at each time-step and outputs a torque. When the arm reaches the limit $\theta = \pm\pi/2$, all energy is absorbed ($\omega = 0$). Euler integration was used with a time-step of 0.01s .

In the experiments, we compare the networks found by Algorithm 1 with two optimal control strategies. The first

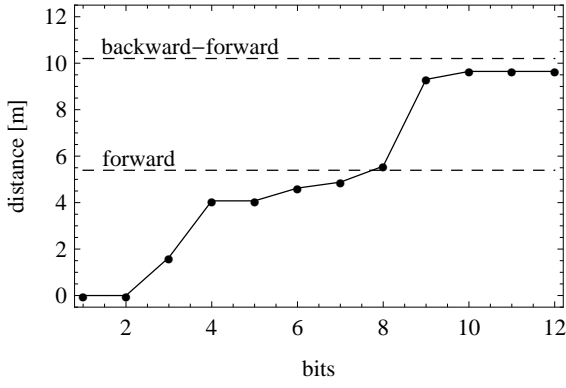


Figure 4: **Ball throwing experiment.** The figure plots the distance reached by the thrown ball against the number of bits, b , used to encode the corresponding two-neuron FRNN weights. Each datapoint denotes the best solution for a given b . The network with weights described with four bits already swings the arm forward and releases the ball with near optimal timing. The network described with eight bits surpasses the optimal forward swing strategy by using a slight backward swing. Nine bits produce a network which swings backward and forward and releases the ball at nearly optimal time. The distances for the two optimal strategies are marked with dashed lines.

applies the highest torque to swing the arm forward, and releases the ball at the optimal angle (which is slightly below 45 degrees, because the ball is always released above the ground). The second, more sophisticated, strategy first applies a negative torque to swing the arm backwards up to the maximum angle, and then applies a positive torque to swing the arm forward, and release the ball at the optimal angle of 43.03 degrees. The optimal distances are 5.391m for the forward swing strategy, and 10.202m for the backward-forward swing strategy.

The results are summarized in Table 3. In 70 evaluations, UNS finds a network with four single-bit coefficients, described by a total of 11 bits, that can throw the ball within almost meter of the optimal forward-swing distance. A more complex 17-bit network is found at evaluation 2516 that uses a slight backward-forward strategy to cross the 5.391m boundary. And finally after 5804 evaluations, a 20-bit network is found that implements a nearly optimal backward-forward swing strategy. Figure 4 shows graphically how the performance progresses as the number of bits, b , representing the coefficient values is increased.

Discussion and Future Directions

The experimental results revealed that, using our approach, the solution networks to some widely used control learning benchmarks are actually quite simple, requiring very short descriptions. However, the question remains whether or not the compressed representation improves search efficiency? In order to quantify the advantage gained by searching for weights indirectly in coefficient space, we compared the performance of random search in weight space against random

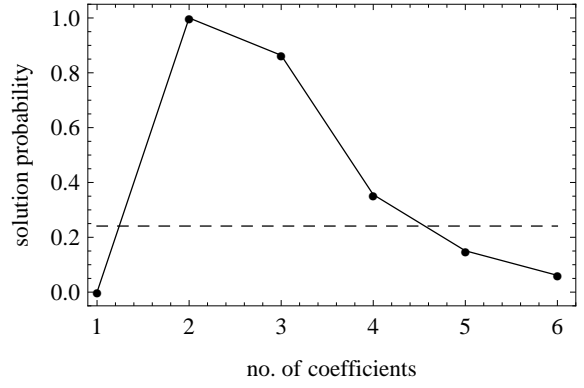


Figure 5: **Coefficient search vs. direct weight search.** The curve shows the probability of finding a solution to the Markov single-pole balancing task within 100 random samples of coefficient space, defined by different numbers of coefficients (calculated of over 1000 runs). The horizontal dashed line at 0.24 indicates the probability of finding a solution by sampling the 6-dimensional weight space directly. Searching for coefficients is more reliable searching weights, for this task, when the number of coefficients is less than five.

search in DCT coefficient space.

Figure 5 shows the results for this comparison in the Markovian single-pole balancing task. Each data-point denotes the probability of finding a successful six-weight neural network (the same architecture that solved the task in Table 1) within 100 random samples, for each number of coefficients. The dashed horizontal line indicates the probability ($p = 0.24$) of finding such a network by randomly sampling the six-dimensional weight space directly. A network represented by just one coefficient is too simple (all weights are equal), and cannot solve the task ($p = 0$). For two, and three coefficients, the task is solved very reliably ($p > 0.9$). As the dimensionality of the coefficient space approaches that of the weights, most of the sampled weight matrices are unnecessarily complex, and, consequently, the probability of finding a solution at random declines rapidly, and falls below the baseline for five and six coefficients (i.e. no compression). This result shows that, on this particular task, just searching the frequency domain without compression only makes the problem harder. It remains to be seen whether compression has a similar profile for all problems, such that there is a *sweet-spot* in a number of coefficients necessary to represent a successful network. However, it seems plausible that, just as with natural signals (e.g. images, video, sound, etc.) most of the energy in useful weight matrices is concentrated in the low frequencies.

In these preliminary experiments, we have focused on benchmark problems for which small network solutions are known to be sufficient. And we have made the implicit assumption that such solutions will have spatially correlated weights. It is possible that, for each task examined here, there exists a permutation in the weight ordering for which the only solutions are those with spatially uncorre-

lated weights, i.e. requiring the full set of coefficients. However, we have made no attempt to predefine amenable weight orderings, and, ultimately, the potential of this approach lies in providing compact representations for large networks, such as those required for vision, where many thousands of inputs have a natural, highly correlated ordering.

In the current implementation, input, recurrent, and bias weights are all combined in a single matrix. For networks with more layers, it may be desirable to specify a separate set of coefficients for each layer, so that the complexity of each matrix can be controlled independently. Also, the way that bits are currently allocated to each coefficient may be too restrictive. A better approach might be to search all partitionings of the bit-string b , instead of roughly according to $(b \bmod c)$, such that the precision of each coefficient is less uniform. For example, fewer bits could be assigned to the lowest frequencies, thereby freeing up more bits for the higher frequencies where more resolution may be needed.

The Universal Network Search algorithm was motivated, in part, by the goal of measuring the complexity of well-known test problems by finding minimal solutions, and made possible because of the small number of bits required to encode the DCT representation. While there is a practical limit on number of bits that can be searched exhaustively (e.g. 32), any, more scalable, optimization method can be applied to search larger numbers of coefficients. Immediate future work will use the indirect DCT network representation in conjunction with evolutionary methods to grow large-scale networks vision-capable robots.

Acknowledgments

The research was supported by the STIFF EU Project (FP7-ICT-231576) and partially by the Humanobs EU Project (FP7-ICT-231453). The authors would like to thank Tom Schaul for extensive consultations.

References

- [BKS09] Zdeněk Buk, Jan Koutník, and Miroslav Šnork. NEAT in HyperNEAT substituted with genetic programming. In *International Conference on Adaptive and Natural Computing Algorithms (ICANNGA 2009)*, 2009.
- [DDWA91] S. Dominic, R. Das, D. Whitley, and C. Anderson. Genetic reinforcement learning for neural networks. In *Proceedings of the International Joint Conference on Neural Networks* (Seattle, WA), pages 71–76. Piscataway, NJ: IEEE, 1991.
- [Gru92] Frederic Gruau. Cellular encoding of genetic neural networks. Technical Report RR-92-21, Ecole Normale Supérieure de Lyon, Institut IMAG, Lyon, France, 1992.
- [GS07] Jason Gauci and Kenneth Stanley. Generating large-scale neural networks through discovering geometric regularities. In *Proceedings of the Conference on Genetic and Evolutionary Computation*, pages 997–1004, New York, NY, USA, 2007. ACM.
- [Kol65] A. N. Kolmogorov. Three approaches to the quantitative definition of information. *Problems of Information Transmission*, 1:1–11, 1965.
- [Lev73] L. A. Levin. Universal sequential search problems. *Problems of Information Transmission*, 9(3):265–266, 1973.
- [LV97] M. Li and P. M. B. Vitányi. *An Introduction to Kolmogorov Complexity and its Applications (2nd edition)*. Springer, 1997.
- [Rad93] Nicholas J. Radcliffe. Genetic set recombination and its application to neural network topology optimisation. *Neural Computing and Applications*, 1(1):67–90, 1993.
- [Ris78] J. Rissanen. Modeling by shortest data description. *Automatica*, 14:465–471, 1978.
- [Sch95] J. Schmidhuber. Discovering solutions with low Kolmogorov complexity and high generalization capability. In A. Prieditis and S. Russell, editors, *Proceedings of the Twelfth International Conference on Machine Learning (ICML)*, pages 488–496. Morgan Kaufmann Publishers, San Francisco, CA, 1995.
- [Sch97] J. Schmidhuber. Discovering neural nets with low Kolmogorov complexity and high generalization capability. *Neural Networks*, 10(5):857–873, 1997.
- [Sch02] J. Schmidhuber. The Speed Prior: a new simplicity measure yielding near-optimal computable predictions. In J. Kivinen and R. H. Sloan, editors, *Proceedings of the 15th Annual Conference on Computational Learning Theory (COLT 2002)*, Lecture Notes in Artificial Intelligence, pages 216–228. Springer, Sydney, Australia, 2002.
- [Sol64] R. J. Solomonoff. A formal theory of inductive inference. Part I. *Information and Control*, 7:1–22, 1964.
- [SS10] Tom Schaul and Jürgen Schmidhuber. Towards a practical universal search. In *Submitted to the Third Conference on Artificial General Intelligence*, 2010.
- [WB68] C. S. Wallace and D. M. Boulton. An information theoretic measure for classification. *Computer Journal*, 11(2):185–194, 1968.
- [Wie91] Alexis Wieland. Evolving neural network controllers for unstable systems. In *Proceedings of the International Joint Conference on Neural Networks* (Seattle, WA), pages 667–673. Piscataway, NJ: IEEE, 1991.

Remarks on the Meaning of Analogical Relations

Ulf Krumnack¹ and Helmar Gust¹ and Angela Schwering² and Kai-Uwe Kühnberger¹

¹ Institute of Cognitive Science
University of Osnabrück, Germany
{krumnack, hgust, kkuehnbe}@uos.de

² Institute of Geoinformatics
University of Münster
schwering@uni-muenster.de

Abstract

Analogical reasoning plays an important role in the context of higher cognitive abilities of humans. Analogies can be used not only to explain reasoning abilities of humans, but also to explain learning from sparse data, creative problem solving, abstractions of concrete situations, and recognition of formerly unseen situations, just to mention some examples. Research in AI and cognitive science has been proposing several different models of analogy making. Nevertheless, no approach for a model theoretic semantics of analogy making is currently available. This paper gives an analysis of the meaning (the semantics) of analogical relations that are computed by the analogy engine HDTP (Heuristic-Driven Theory Projection).

Introduction

Humans show remarkable higher cognitive abilities comprising not only (abstract) types of reasoning, learning from sparse data, and planning, but also the ability to creatively finding new conceptualizations of an unknown domain, to solve problems based on similar solutions for other problems, and to recognize and categorize perceptual input that was never seen before. At least to a certain extent it is possible to explain and to model such types of higher cognitive abilities with frameworks for analogical reasoning (GHK01).

Because of the wide range of applicability of analogy making mechanisms for the explanation of various higher cognitive abilities, analogies are of interest not only for artificial intelligence and cognitive science, but also for artificial general intelligence. Classical AI methodologies suffer from various deficiencies that result from highly specialized models. For example, it is hard to find uniform frameworks for the variety of different reasoning types (GKSK09), for creativity aspects of human cognition (Ind92), or for problem solving in unknown situations (GP07). As a consequence of these deficiencies the desired generalization capabilities of appropriate AGI systems are currently far from being reachable. To tackle this problem we propose the usage of analogy mechanisms for AGI systems.

Frameworks for analogy making cover a large part of cognitive abilities that are usually considered to be

central for AGI systems. A good source to support this claim is (KHG09) where analogies are used to model aspects of reasoning, creativity, problem solving, learning, perception, or motor control, just to mention some of them. Furthermore, even from a more abstract perspective the establishment of analogical relations seems to be one of the rare possibilities to explain many cognitive phenomena in a uniform way: quite often we act (alternatively perceive, reason, learn etc.) as if we were in another (well-known and analogous) situation. It rarely happens that humans can reason in a purely deductive (abductive, inductive etc.) way to act in real life. A natural description of such cognitive phenomena can be provided by analogies, because vagueness, learning, and the transfer of knowledge about old situations to new ones are intrinsically embedded in the very idea of analogy making.

Due to the fact that analogies are considered to be a central mechanism of human cognition and intelligence, a number of models have been proposed to explain different aspects of analogies, varying in complexity and in their degree of formalization. A few examples of such frameworks are SME (FFG89), interactionism (DIS03), LISA (HH96), Copycat (Hof95), and AMBR / DUAL (KP01).¹ Most of these analogy making models use special means for representation and computation of analogical inferences. For example, the structure mapping engine (FFG89), probably the currently best known analogy model, uses a graph-based representation and a heuristic-governed matching algorithm.

Although, we can conclude that there is a broad variety of different computational models, it is hard to find a spelled out semantics of analogical relations computed by algorithmic models. Even worse, for all established frameworks even an endeavor to find a semantics of the underlying computations cannot be found. However, from an AGI perspective, it would be desirable to have a model that could be combined with standard mechanisms of knowledge representation and reasoning.

¹ A good overview of various theoretical and practical approaches for the modeling of analogy making can be furthermore found in the special issue of *Cognitive Systems Research* about analogical reasoning as a means for the integration of cognitive abilities (SKK09).

In this paper, we will discuss heuristic-driven theory projection (HDTP), a formal framework to compute analogies between domains that are described by classical first-order formulas (GKS06). HDTP’s matching algorithm is based on the syntactic representation and analogies are computed as mappings of formula sets induced by their domain axiomatizations (SKKG09). This approach fits into the classical paradigm of symbol processing, where operations are performed on a symbolic level in a way that a coherent semantic interpretation can be provided. However, from this perspective, the framework remains incomplete, until a semantic characterization of the syntactic operations is provided.

In this paper, we will show, that the syntactic mapping procedure of HDTP can be given a sensible interpretation on the semantic side. Not only, the syntactic mapping can be shown to induce a mapping between models (this was already shown in (GKKS07)), but furthermore, the generalized formulas constructed by HDTP during the mapping process can be interpreted as a new, abstract domain theory, which can be given a model theoretic semantics. To our knowledge this is the first concrete approach to make the semantics of analogy making formally precise.

The paper has the following structure. First, we will sketch the syntactic basis of HDTP and the challenges of developing a semantics for HDTP, then we will sketch the theory of institutions. We will continue with describing an institution theoretic analysis of analogy making. Finally, we will sketch an example and we will add some concluding remarks.

Heuristic-Driven Theory Projection

The Theory in a Nutshell

HDTP is a framework for discovering analogous structures of pairs of logical theories. In this section, we sketch the main ideas of the syntactic anti-unification process of HDTP. For a thorough introduction to the syntactic principles of HDTP, the reader is referred to (SKKG09).

- Two sets of formulas Ax_S and Ax_T are provided as input specifying an axiomatization of the source and target domain, respectively. Each axiomatization describes (some aspects) of a *domain* of knowledge. These sets of formulas Ax_S and Ax_T induce corresponding theories Th_S and Th_T (i.e. the deductive closure of Ax_S and Ax_T).
- A pair of clauses $c_S \in Th_S$ and $c_T \in Th_T$ is selected and (syntactically) generalized by anti-unification to a clause c_G .² For generalization, domain symbols

- can be kept, if they occur in both domains (e.g. a relation symbol $>$)
- can be generalized into a new variable, e.g. *sun*, *nucleus* are generalized to an individual variable X in the famous Rutherford analogy describing the analogy between the solar system and the atom model
- can be dropped by an “argument insertion” substitution, i.e. symbols are “integrated” into a complex function.

As a by-product, the computed generalization provides a pair of substitutions $\langle \sigma, \tau \rangle$ with $c_G \sigma = c_S$ and $c_G \tau = c_T$. In other words, the source and target clauses c_S and c_T can be gained by applying the substitutions to the corresponding generalized clause c_G , respectively.

- A set Ax_G of generalized clauses is incrementally built by repeating the described generalization step. This set is considered to be good, if it has a low substitution complexity and a high degree of coverage³
- The substitutions belonging to a (good) set of generalizations can be used to establish an analogical relation
- Based on this relation, formulas of the source domain that are not anti-unified yet can be transferred (projected) to the target domain in order to allow new conceptualizations of the target.

Challenges

There are several challenges of this approach if one wants to develop a model theoretic semantics for HDTP. First, the generalized expression Ax_G are not necessarily first-order logical formulas (FOL formulas). The anti-unification process may result in a second-order generalization, for example, by introducing a relation or function variable. Second, the variables introduced by anti-unification might be treated differently than ordinary (universally quantified) variables. In particular, second-order variables can be interpreted as existentially quantified in order to prevent inconsistencies or they need to get a treatment as described below. Third, the question arises how an adequate notion for a “model” (in the logical model theoretic sense) of the resulting analogy might look like.

Here are the basic ideas of the present approach: In (GKS06), it is proposed, that the generalized terms can be seen as axioms of a generalized theory. A framework is sketched that integrates a semantics for the analogical relation and the generalized theory. However, some questions are left open and especially the status of the variables introduced by anti-unification remains

²Anti-unification can be understood as the dual construction of unification, i.e. instead of computing the most general unifier, the most specific (least general) generalization of terms and formulas is computed. Anti-unification was introduced by Plotkin (Pl070).

³Coverage is the concept that “measures” the degree of specificity of a generalization. Intuitively, we can say that an anti-unifier $\langle Ax_G, \sigma, \tau \rangle$ has at least the same coverage as the anti-unifier $\langle Ax'_G, \sigma', \tau' \rangle$ if there exists a substitution $\Theta : Ax'_G \rightarrow Ax_G$, such that $\sigma' = \sigma \circ \Theta$ and $\tau' = \tau \circ \Theta$. Compare (SKKG09) for the details of this concept.

unclear. If these generalized symbols would be considered as logical variables, the resulting formulas leave the realm of first-order logic, as anti-unification can also generalize function and predicate symbols. Therefore, we propose to see the generalized symbols not as variables, but as elements of a new vocabulary. The generalized theory is then a classical first-order theory with formulas built from these symbols. The substitutions induce a mapping between the generalized theory and the domain theories.

An Institutional View on Analogies

To make these ideas more precise, we will use the language of the theory of institutions (GB92).⁴ Institutions provide a framework to describe model theory at an abstract level using methods from category theory. Informally, an institution consists of a collection of signatures **Sign** and to each signature Σ , first the collection **Sen**(Σ) of all Σ -sentences is assigned. In the case of FOL, the Σ -sentences correspond to the set of all FOL formulas that can be built using symbols from Σ . Second, for each signature Σ the collection **Mod**(Σ) of all Σ -models is assigned. In the case of FOL, this collection correspond to all possible interpretations of symbols from Σ . The Σ -models and Σ -sentences are related by the relation of Σ -satisfaction, usually denoted by \models_Σ .

A central idea of the theory of institutions is, that truth is invariant under the change of notation. Within an institution, signatures can be mapped to other signatures by so called signature morphisms, i.e. structure preserving functions. In FOL, this translates into the constraint that arity and sortal restrictions have to match. A signature morphism $f : \Sigma \rightarrow \Sigma'$ induces functions **Sen**(f) : **Sen**(Σ) \rightarrow **Sen**(Σ') and **Mod**(f) : **Mod**(Σ') \rightarrow **Mod**(Σ) which have to be consistent with Σ -satisfaction as specified by the following contravariant satisfaction constraint: for all $\varphi \in \mathbf{Sen}(\Sigma)$ and $M' \in \mathbf{Mod}(\Sigma')$:

$$M' \models_\Sigma \mathbf{Sen}(f)(\varphi) \Leftrightarrow \mathbf{Mod}(f)(M') \models_\Sigma \varphi$$

Every collection T of Σ -sentences (called Σ -theory) determines a class of Σ -models:

$$T^* = \{M \mid M \models_\Sigma \varphi \text{ for all } \varphi \in T\}$$

Dually, every collection \mathcal{V} of Σ -models determines a Σ -theory:

$$\mathcal{V}^* = \{\varphi \mid M \models_\Sigma \varphi \text{ for all } M \in \mathcal{V}\}$$

As a natural consequence we get that the double application of these operators provides the closure under semantic entailment: $T^\bullet = T^{**}$.

⁴For a thorough discussion of logic systems in the language of the theory of institutions the reader is referred to the monograph (Dia08).

The Generalized Theory

We now describe HDTP within the framework of institutions, more precisely, within the institution FOL. The new variables introduced by the process of anti-unification are placeholders, which are instantiated by different terms on the source and target side. In what follows, we will treat these variables as new symbols which can be used as a signature for a generalized theory of both domains. Using this idea we can give the intuitive notion of “generalized theory” a formal meaning.

The signature Σ_G will consist of all symbols used in the generalized formulas Ax_G , i.e. symbols from $\Sigma_S \cap \Sigma_T$ and the variables introduced by anti-unification. The generalized formulas Ax_G are then ordinary first-order formulas over the signature Σ_G in the usual sense. The theory spanned by these formulas will be called (as above) the generalized theory $Th_{\Sigma_G}(Ax_G)$.⁵

The generalized axioms Ax_G are related to the axiomatizations Ax_S and Ax_T of the source and target domain by the following relations

$$Ax_G \sigma \subseteq (Ax_S)^\bullet \quad \text{and} \quad Ax_G \tau \subseteq (Ax_T)^\bullet$$

with σ and τ being the substitutions computed by HDTP. The application of the substitutions σ and τ defines mappings $\sigma^* : \mathbf{Sen}(\Sigma_G) \rightarrow \mathbf{Sen}(\Sigma_S)$ and $\tau^* : \mathbf{Sen}(\Sigma_G) \rightarrow \mathbf{Sen}(\Sigma_T)$. It can be shown that these mappings can be restricted to the respective theories, i.e. that $\sigma^*| : (Ax_G)^\bullet \rightarrow (Ax_S)^\bullet$ and $\tau^*| : (Ax_G)^\bullet \rightarrow (Ax_T)^\bullet$. However, in general the induced mappings are not theory morphisms in the sense of (GB92), as there are no underlying signature morphisms. To overcome this problem, we will extend the original domain signatures Σ_S and Σ_T .

Substitutions

In the setting of HDTP, there exists a set of (higher-order) generalized formulas, which can be instantiated via substitution to domain formulas over given signatures. We will describe this situation in terms of signatures and signature morphisms.

Let Ax be a set of (higher-order) formulas and θ be a substitution such that $Ax\theta$ is a set of (first-order) formulas over some signature Σ . Now we can define a signature Σ_θ which allows to interpret the formulas Ax as first-order formulas as follows:

- (i) We extend Σ by taking into account the pairs X/t in θ and by introducing a new symbol X for each such pair. Notice, that there is an obvious embedding $\Sigma \hookrightarrow \Sigma_\theta$.
- (ii) A system of equations and equivalences is used to relate new symbols in Σ_θ to those in Σ .

⁵From now on we will use $Th_\Sigma(Ax)$ to denote the closure under semantic entailment of an axiomatization Ax restricted to sentences over a signature Σ , i.e. $Th_\Sigma(Ax) = (Ax)^\bullet \cap \mathbf{Sen}(\Sigma)$.

Consider again a substitution θ as above. We can define a set of Σ_θ -formulas Def_θ as follows:⁶

Pair in substitution θ	Formula in Def_θ
X/c	$X = X\theta$
$F/f_1 \circ \dots \circ f_n$	$\forall \bar{x} F(\bar{x}) = F\theta(\bar{x})$
P/p	$P \leftrightarrow p$
$P/p(f_1 \circ \dots \circ f_n)$	$\forall \bar{x} P(\bar{x}) \leftrightarrow P\theta(\bar{x})$

The idea is, that Def_θ provides a description that allows the application of substitutions as a logical deduction in the HDTP algorithm. One can make this more precise by noticing the following fact: If Ax, θ, Σ are given as above, then the following two facts hold:

- $Th_{\Sigma_\theta}(Ax \cup Def_\theta) \cap \mathbf{Sen}(\Sigma) = Th_\Sigma(Ax\theta)$.
- There is a (natural) isomorphism $Mod_{\Sigma_\theta}(Ax \cup Def_\theta) \cong Mod_\Sigma(Ax\theta)$.

The crucial point is, that the additional newly introduced symbols do not allow to derive new sentences in $\mathbf{Sen}(\Sigma)$. Notice that they cannot introduce interrelations between existing symbols provided that these additional symbols are really new. On the semantic level, the additional symbols do not provide new models, as the interpretations of these symbols are fully constrained by the formulas from Def_θ .

Coverage

Usually, an analogy does not establish a complete mapping between two domains. Quite often only parts of the domains are associated with each other. This fact can be used to motivate the concept of coverage in HDTP. A domain formula φ from $\mathbf{Sen}(\Sigma)$ is said to be covered by an analogy iff $\varphi \in Th(Ax_G\theta)$, i.e. ϕ is either an instantiation of a generalized formula or can be derived from such instantiations. The set $Th(Ax_G\theta)$ is called the part of the domain that is covered by the generalization (or analogy).

Assume that Ax_D is a set of domain axioms over some signature Σ , Ax_G a set of generalized formulas, θ a substitution that instantiates Ax_G in $\mathbf{Sen}(\Sigma)$, i.e. $Ax_G\theta \subseteq Th_\Sigma(Ax_D)$. Then one can easily observe that the following relations (i) - (iv) hold:

- (i) $Th_\Sigma(Ax_G\theta) \subseteq Th_\Sigma(Ax_D)$
- (ii) $Th_{\Sigma_\theta}(Ax_G \cup Def_\theta) \cap \mathbf{Sen}(\Sigma) = Th_\Sigma(Ax_G\theta)$
- (iii) $Th_{\Sigma_\theta}(Ax_G \cup Def_\theta) \cap \mathbf{Sen}(\Sigma) \subseteq Th_\Sigma(Ax_D)$
- (iv) $Th_{\Sigma_\theta}(Ax_G \cup Def_\theta \cup Ax_D) \cap \mathbf{Sen}(\Sigma) = Th_\Sigma(Ax_D)$

The first formula just states that all covered formulas are domain formulas, a necessary condition to call Ax_G

⁶The Σ_θ -formulas Def_θ are naturally induced by the theory of “restricted higher-order anti-unification” as described in (SKKG09).

a generalization of the domain. The second formula describes the covered part of the domain theory as a (subset of) the closure under semantic entailment of $Ax_G \cup Def_\theta$. The third formula is a trivial consequence from (i) and (ii). The fourth formula states that adding $Ax_G \cup Def_\theta$ does not change the set of entailed formulas in $\mathbf{Sen}(\Sigma)$. It is an immediate consequence of (iii).

Using the contravariant satisfaction constraint of the theory of institutions, we can characterize the corresponding model classes as dual statements of the above results as follows:

- (i) $\mathbf{Mod}_\Sigma(Ax_G\theta) \supseteq \mathbf{Mod}_\Sigma(Ax_D)$
- (ii) $\mathbf{Mod}_{\Sigma_\theta}(Ax_G \cup Def_\theta) \cong \mathbf{Mod}_\Sigma(Ax_G\theta)$
- (iii) $\mathbf{Mod}_{\Sigma_\theta}(Ax_G \cup Def_\theta) \hookrightarrow \mathbf{Mod}_\Sigma(Ax_D)$ is injective.
- (iv) $\mathbf{Mod}_{\Sigma_\theta}(Ax_G \cup Def_\theta \cup Ax_D) \cong \mathbf{Mod}_\Sigma(Ax_D)$

The crucial point is again, that we have no choice when interpreting the new symbols, as their value is determined by the defining equations from Def_θ .

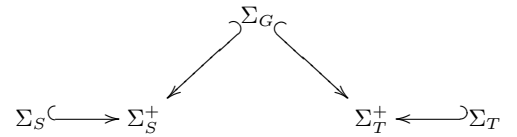
Describing Analogies

We will now turn to analogies and their description in HDTP. The starting point is given by two domain axiomatizations Ax_S and Ax_T over signature Σ_S and Σ_T , respectively. HDTP will then compute a set of generalized formulas Ax_G and substitutions σ, τ such that $Th_{\Sigma_S}(Ax_G\sigma) \subseteq Th_{\Sigma_S}(Ax_S)$ and $Th_{\Sigma_T}(Ax_G\tau) \subseteq Th_{\Sigma_T}(Ax_T)$. Following the HDTP process described above we can construct signatures $\Sigma_{S,\sigma}$ and $\Sigma_{T,\tau}$ such that Ax_G are first-order formulas over these signatures. We set $\Sigma_G = \Sigma_{S,\sigma} \cap \Sigma_{T,\tau}$ and conclude that $Ax_G \subseteq \mathbf{Sen}(\Sigma_G)$.

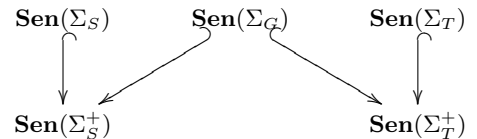
For the following discussion we introduce some abbreviations:

$$\Sigma_S^+ := \Sigma_S \cup \Sigma_G \quad \text{and} \quad \Sigma_T^+ := \Sigma_T \cup \Sigma_G$$

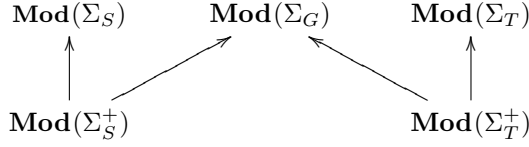
Then the sets of equations Def_σ and Def_τ are formulas over Σ_S^+ and Σ_T^+ , respectively. Furthermore, we will set $Ax_S^+ := Ax_S \cup Def_\sigma$ and $Ax_T^+ := Ax_T \cup Def_\tau$. We have the following inclusion relations between signatures:



These signature inclusions induce inclusions on the syntactic level (notice that we work in the institution FOL):



Dually the above signature inclusions induce the inverse inclusions on the semantic level:



We will now examine the subsets of these sets, that are associated with the domain theories. Compare Figure 1 for these subset relations. The claim that i_S and i_T are inclusions, follows from

$$(Ax_S^+ \cup Ax_G)^{**} = (Ax_S^+)^{**}$$

On the semantic level \tilde{j}_S and \tilde{j}_T are isomorphisms (even though j_S and j_T are not), while \tilde{i}_S and \tilde{i}_T are in general neither injective nor surjective.

Analogical Relation

Given the considerations so far, we can define the following mappings (cf. Figure 1):

$$m_S : (Ax_S)^* \rightarrow (Ax_G)^* \quad \text{and} \\ m_T : (Ax_T)^* \rightarrow (Ax_G)^*$$

by $m_S = \tilde{i}_S \circ \tilde{j}_S^{-1}$ and $m_T = \tilde{i}_T \circ \tilde{j}_T^{-1}$. Given models $M_S \in (Ax_S)$ and $M_T \in (Ax_T)$, for every symbol $x \in \Sigma_G$ we get two interpretations $m_S(M_S)(x) \in U_S$ and $m_T(M_T)(x) \in U_T$. Associating those elements establishes the analogical relation. Briefly, this is the idea to relate $M_S(x\sigma) \sim M_T(x\tau)$ which corresponds to the relation $x\sigma \sim x\tau$ on the syntactic level. Clearly, it is possible to extend this association to terms and even formulas.

Example: Heat Flow

It might be worth to spell out these ideas in an example. We have chosen the heat flow analogy, which models the creation of new concepts “heat” and “heat flow” by analogical inference, given a model of water flow as the source domain. In the context of HDTP, this analogy seems to be especially interesting as it makes some complex mappings necessary. For a detailed discussion of the computation compare (SKKG09)).

We will follow (GB92) and describe signatures by two sets: function symbols and predicate symbols, both annotated with arity. So we start with the following domain signatures and axiomatizations:

$$\begin{aligned} \Sigma_S &= \{ \{height/2, in/2, water/0, beaker/0, vial/0, \\ &\quad t_1/0, t_2/0\}, \{>/2\} \} \\ Ax_S &= \{ height(water \text{ in } beaker, t_1) > \\ &\quad height(water \text{ in } vial, t_1) \rightarrow (t_1 > t_2) \\ &\quad \wedge height(water \text{ in } beaker, t_1) > \\ &\quad \quad height(water \text{ in } beaker, t_2) \\ &\quad \wedge height(water \text{ in } vial, t_2) > \\ &\quad \quad height(water \text{ in } vial, t_1), \\ &\quad \dots \} \end{aligned}$$

The signatures Σ_S of the source domain provides the vocabulary to describe the flow of water in two connected vessels, a beaker and a vial. Due to space restriction we have only depicted one particular fact from the axiomatization: if the water level in the beaker is higher than the water level in the vial at a given time point t_1 , then for every subsequent observation the water level will decline in the beaker while it will increase in the vial. For the target domain we provide the following axiomatization:

$$\begin{aligned} \Sigma_T &= \{ \{temp/2, in/2, coffee/0, cup/0, cube/0, \\ &\quad t_1/0, t_2/0\}, \{>/2\} \} \\ Ax_T &= \{ temp(coffee \text{ in } cup, t_1) > temp(cube, t_1) \\ &\quad \rightarrow (t_1 > t_2) \\ &\quad \wedge temp(coffee \text{ in } cup, t_1) > \\ &\quad \quad temp(coffee \text{ in } cup, t_2) \\ &\quad \wedge temp(cube, t_2) > temp(cube, t_1), \\ &\quad \dots \} \end{aligned}$$

The vocabulary provides elements to describe a situation in which a cup of hot coffee, when connected with an cold ice cube, will cool down, while the ice warms up. From this axiomatization, HDTP will compute the set of generalized formulas:

$$\begin{aligned} Ax_G &= \{ E(X, t_1) > E(Y, t_1) \\ &\quad \rightarrow (t_1 > t_2) \\ &\quad \wedge E(X, t_1) > E(X, t_2) \\ &\quad \wedge E(Y, t_2) > E(Y, t_1), \\ &\quad \dots \} \end{aligned}$$

and substitutions

$$\begin{aligned} \sigma &= \{E/\lambda u, v.height(water \text{ in } u, v), X/beaker, Y/vial\} \\ \tau &= \{E/\lambda u, v.temp(u, v), X/coffee \text{ in } cup, Y/cube\} \end{aligned}$$

The formulas in Ax_G are (higher-order) formulas over the signature

$$\Sigma_S \cap \Sigma_T = \{ \{t_1/0, t_2/0\}, \{>/2\} \}$$

with generalization variables E, X, Y . Now, interpreting these variables (that occur in the domain substitutions) as new symbols, we get an extended signature:

$$\Sigma_G = \{ \{E/2, X/0, Y/0, t_1/0, t_2/0\}, \{>/2\} \}$$

The defining equations Def_σ and Def_τ are constructed as sets of formulas over the signatures $\Sigma_G^+ \cup \Sigma_S$ and $\Sigma_G^+ \cup \Sigma_T$, respectively:

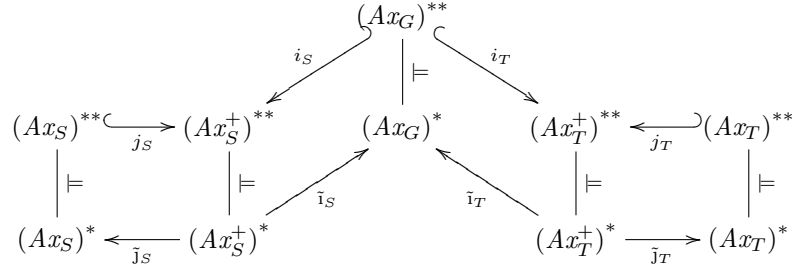


Figure 1: Subset relations that are associated with the domain theories.

$$Def_{\sigma} = \{ \quad \forall u, v : E(u, v) = height(water\ in\ u, v), \\ X = beaker, Y = vial \}$$

$$Def_{\tau} = \{ \quad \forall u, v : E(u, v) = temp(u, v), \\ X = coffee\ in\ cup, Y = cube \}$$

As can be seen from these formulas, the complex expression *height(water in u, v)* on the source side corresponds to the less complex term *temp(u, v)* on the target side, while the complex expression *coffee in cup* on the target corresponds to the constant *beaker* on the source.

Conclusions

In this paper, we propose that analogy making can be used for various applications in the AGI context. We sketched the framework HDTP for analogy making that is intended to compute a generalized theory of given input source and target domains. The main contribution of this paper is a new institution-based formal semantics of the established analogical relation together with the model theory of the computed generalized theory. A classical example of analogy making is also roughly sketched. Future work will be a complete specification of the model theory of analogical relations, a representative set of worked out examples, and the discussion of potential alternatives to the present approach.

References

- R. Diaconescu. *Institution-Independent Model Theory*. Studies in Universal Logic. Birkhäuser, 2008.
- M. Dastani, B. Indurkha, and R. Scha. An algebraic approach to modeling analogical projection in pattern perception. *Journal of Experimental and Theoretical Artificial Intelligence*, 15(4):489–511, 2003.
- B. Falkenhainer, K. Forbus, and D. Gentner. The structure-mapping engine: Algorithm and examples. *Artificial Intelligence*, 41(1):1–63, 1989.
- J. Goguen and R. Burstall. Institutions: Abstract model theory for specification and programming. *Journal of the ACM*, 39(1):95–146, 1992.
- D. Gentner, K. Holyoak, and B. Kokinov, editors. *The Analogical Mind. Perspectives from Cognitive Science*. MIT Press, Cambridge, Mass, 2001.

H. Gust, U. Krumnack, K.-U. Kühnberger, and A. Schwering. An approach to the semantics of analogical relations. In *Proceedings of the EuroCogSci 2007*, 2007.

H. Gust, K.-U. Kühnberger, and U. Schmid. Metaphors and heuristic-driven theory projection. *Theoretical Computer Science*, 354(1):98–117, 2006.

H. Gust, U. Krumnack, A. Schwering, and K.-U. Kühnberger. The role of logic in AGI systems: Towards a Lingua Franca for general intelligence. In *Proceedings of the Second Conference of Artificial General Intelligence*, 2009.

B. Goertzel and C. Pennachin. Preface. In B. Goertzel and C. Pennachin, editors, *Artificial General Intelligence*. Springer, 2007.

J. Hummel and K. Holyoak. LISA: A computational model of analogical inference and schema induction. In *Proceedings of 16th Meeting of the Cognitive Science Society*, 1996.

D. Hofstadter. *Fluid Concepts and Creative Analogies*. Basic Books, 1995.

B. Indurkha. *Metaphor and Cognition*. Kluwer Academic Publishers, Dordrecht, 1992.

B. Kokinov, K. Holyoak, and D. Gentner, editors. *Proceedings of the Second International Analogy Conference – Analogy 09*, New Frontiers in Analogy Research. NBU Press, 2009.

B. Kokinov and A. Petrov. Integrating memory and reasoning in analogy-making: The AMBR model. In D. Gentner, K. Holyoak, and B. Kokinov, editors, *The Analogical Mind. Perspectives from Cognitive Science*, pages 59–124. MIT Press, 2001.

G. Plotkin. A note on inductive generalization. *Machine Intelligence*, 5:153–163, 1970.

A. Schwering, K.-U. Kühnberger, and B. Kokinov, editors. *Special Issue on Analogies - Integrating Cognitive Abilities*, volume 10 of *Cognitive Systems Research*. Elsevier, 2009.

A. Schwering, U. Krumnack, K.-U. Kühnberger, and H. Gust. Syntactic principles of heuristic-driven theory projection. *Cognitive Systems Research*, 10(3):251–269, 2009.

Efficient Constraint-Satisfaction in Domains with Time

Perrin G. Bignoli, Nicholas L. Cassimatis, Arthi Murugesan

Department of Cognitive Science
Rensselaer Polytechnic Institute
Troy, NY 12810

{bignop, cassin, muruga}@rpi.edu

Abstract

Satisfiability (SAT) testing methods have been used effectively in many inference, planning and constraint satisfaction tasks and thus have been considered a contribution towards artificial general intelligence. However, since SAT constraints are defined over atomic propositions, domains with state variables that change over time can lead to extremely large search spaces. This poses both memory- and time-efficiency problems for existing SAT algorithms. In this paper, we propose to address these problems by introducing a language that encodes the temporal intervals over which relations occur and an integrated system that satisfies constraints formulated in this language. Temporal intervals are presented as a compressed method of encoding time that results in significantly smaller search spaces. However, intervals cannot be used efficiently without significant modifications to traditional SAT algorithms. Using the Polyscheme cognitive architecture, we created a system that integrates a DPLL-like SAT-solving algorithm with a rule matcher in order to support intervals by allowing new constraints and objects to be lazily instantiated throughout inference. Our system also includes constraint graphs to compactly store information about temporal and identity relationships between objects. In addition, a memory retrieval subsystem was utilized to guide inference towards minimal models in common sense reasoning problems involving time and change. We performed two sets of evaluations to isolate the contributions of the system's individual components. These tests demonstrate that both the ability to add new objects during inference and the use of smart memory retrieval result in a significant increase in performance over pure satisfiability algorithms alone and offer solutions to some problems on a larger scale than what was possible before.

Introduction

Many AI applications have been successfully framed as SAT problems: planning (Kautz and Selman 1999), computer-aided design (Marques-Silva and Sakallah 2000), diagnosis (Smith and Veneris 2005), and scheduling (Feldman and Golumbic 1990). Although SAT-solvers have successfully handled problems with millions of clauses, tasks that require an explicit representation of time can exceed their capacities.

Adding a temporal dimension to a problem space has the potential to greatly expand search space sizes because SAT algorithms propositionalize relational constraints. The most

direct way to incorporate time is to have a copy of each state variable for every time point over which the system reasons. This increases the number of propositions by a factor equal to the number of time points involved. Since SAT algorithms generally become slower as the size of a problem increases, adding a temporal dimension to even relatively simple problems can make them intractable.

Although problems with time require more space to encode, the true expense of introducing time stems from the additional cost required to find a SAT solution. Consider a task that requires the comparison of all the possible ways that a car can visit three locations in order. If the problem has no reference to time points, there is only one solution: the car just moves from *a* to *b* to *c*. On the other hand, there are clearly more possibilities, since the car could potentially move or not move at every time. Compared to other SAT-solvers, LazySAT (Singla and Domingos 2006), which lazily instantiates constraints, is less affected by the increased memory demands of larger search spaces. Unfortunately, lazy instantiation will not increase the tractability of larger problems with respect to runtime.

There is, however, a more efficient way of representing time. Since it is unlikely that the truth value of a proposition will change at every time point, temporal intervals can be used to denote segments of contiguous time points over which its value is constant. This practice alleviates the need to duplicate all propositions at every time point for most problem instances, thus significantly reducing the search space size. Intervals also mitigate the arbitrary granularity of time because they are continuous and scale independent.

However, existing SAT solvers cannot process intervals efficiently because they do not allow new objects to be introduced during the course of inference. It is clearly impossible to know exactly which temporal intervals will be required. Therefore, every possible interval must be defined in advance. Since $\frac{(n-1)(n-2)}{2}$ unique intervals can be defined over *n* times, there would be little advantage to use them with current searching methods.

To capture the benefits of SAT while supporting the use of intervals, we created an integrated system that combines a DPLL-like search with several specialized forms of inference: a rule matcher, constraint graphs, and memory retrieval. Rule matching allows our system to both lazily instantiate constraints and introduce new objects during

inference. Constraint graphs compactly store temporal and identity relationships between objects. Memory retrieval supports common sense reasoning about time. Although SAT has previously been applied to planning (Shanahan and Witkowski 2004) and reasoning (Mueller 2004) with the event calculus, we present a novel approach.

Language

We have specified a language that can express relationships between objects at different points in time. This language incorporates temporal intervals and a mechanism for introducing new objects during inference. For example, in path planning tasks, it is often useful to have a condition such as: If at location $p1$ and must be at location $p2$, which is not adjacent to $p1$, then move to some location px that is adjacent to $p1$ at the next possible time. We write this constraint as:

$$\text{Location}(?x, ?p1, ?t1) \wedge \text{Location}(?x, ?p2, ?t2) \wedge \neg \text{Same}(?p1, ?p2, E) \Rightarrow (\infty) \\ \text{Adjacent}(?p1, ?px, E) \wedge \text{Meets}(?t1, ?tx, E)$$

Note that we prefix arguments with a ‘?’ to denote variables. Variables permit this one constraint to apply to all locations in the system. Additionally, we can assign weights to constraints as a measure of their importance. Because this constraint holds for any path, it is given an infinite weight to indicate that it must be satisfied.

Formally, constraints have the form $A_1 \wedge \dots \wedge A_m \Rightarrow (w) C_1 \wedge \dots \wedge C_n$, where w is a positive rational number, $m \geq 0$ and $n \geq 1$. A_i and C_j are first order literals. Literals have the form $P(\text{arg}_1, \dots, \text{arg}_n)$, where P is a predicate, arg_i is a term, and arg_n must be a time point or interval. Terms that are prefixed with a ‘?’ are variables; others are constant “objects.” A grounded predicate is one in which no term is a variable. Predicates specify relations over the first $n-1$ terms, which hold at time arg_n . There is a special type of relation called an attribute. Attributes are predicates, P , with three terms, o , v , and t , such that only one relation of the form $P(o, v, t)$ holds for each o at every t . The negation of a literal is expressed with the \neg operator. Every literal is mapped to a truth value.

If all of the literals in a constraint are grounded, then the constraint itself is grounded. Only grounded constraints can be satisfied or broken, according to the truth value of its component literals. A constraint is broken iff every antecedent literal is assigned to true and every consequent literal is assigned to false. The cost of breaking a constraint is given by w , which is infinite if the constraint is hard.

Some predicates and objects are included in the language. For instance, Meets, Before, and Includes are temporal relations that are similar to the predicates in Allen’s interval calculus (Allen 1981). We reserve a set of objects of the form $\{t1, \dots, tn\}$, where n is the number of times in the system. This set is known as the native times. Another time, E , denotes eternity and is used in literals whose assignments do not change over time.

A model of a theory in this language consists of a complete assignment, which is a mapping of every literal to a truth value. Valid models are those such that their assignment permits all hard constraints to be satisfied. All models have an associated cost equal to the cost of its broken constraints. Each theory has many valid models, but it is often useful to find one of the models with the minimum cost. For instance, this process can perform change minimization, a form of commonsense reasoning motivated by the frame problem (Shanahan 1997).

System Architecture

We created an integrated system using the Polyscheme cognitive architecture (Cassimatis 2002) in order to efficiently solve problems with time. This approach allowed us to glean the benefits of SAT while capitalizing on the properties of specialized forms of inference. It is easiest to describe how our system works by framing it as a DPLL-like search. DPLL (Davis, Logemann et al. 1962) performs a branch-and-bound depth first search that is guaranteed to be complete for finite search spaces. The algorithm searches for the best assignment by making assumptions about the literals that appear in its constraint set. An assumption consists of selecting an unassigned literal, setting it to true or false, and then performing inference based on this assignment. When necessary, the search backtracks to previous decision points to explore the ramifications of making the opposite assumption.

The DPLL-FIRST procedure in Algorithm 1 takes a set of constraints, c , as input and outputs an assignment of literals to truth values that minimizes the cost of broken constraints. In the input constraint set, there must be at least one fully grounded constraint with a single literal. Such constraints are called facts. Within the DPLL-FIRST procedure, several data structures are declared and passed to DPLL-RECUR, which is illustrated in Algorithm 2. First, there is a structure, *assign*, which stores the current assignment of literals to truth values. The facts specified in the input are stored in *assign* with an assignment that corresponds to the valence of the fact’s literal. Second, there is a queue, q , which stores the literals that have been deemed relevant by the system in order to perform inference after the previous assumption. At the beginning, q contains the facts in the input. Third, b stores the best total assignment that has been found so far. Fourth, the cost of b is stored in o . Initially, b is empty and o is infinite.

Algorithm 1. DPLL-FIRST(c):

return DPLL-RECUR($c, \text{assign}, q, o, b$)

Each time DPLL-RECUR is called, it performs an elaboration step that infers new assignments based on the current assumption. Initially, when there is no assumption, the elaboration step attempts to infer new information from the constraints specified in the input. After elaboration, the current assignment is examined to determine if one of the

three termination conditions is met. The first condition is if the assignment is contradictory because the same literal has been assigned to both true and false. The second condition is if the cost of the current assignment exceeds the lowest cost of a complete assignment that has been discovered in previous iterations. Since new assignments can never reduce the total cost, it is unnecessary to continue searching. The third condition is if the current assignment is complete. In all of these cases, the search backtracks to a previous assumption and investigates any remaining unexplored possibilities. Afterwards, the search selects an unassigned literal and creates two new branches in the search tree: one where the literal is assumed to be true and one where it is assumed to be false. DPLL-RECUR is then invoked on those subtrees and the assignment from the branch with the lower cost is returned.

Algorithm 2. DPLL-RECUR($c, assign, q, o, b$)

```

call ELABORATION( $c, assign, q$ )
if Contradictory( $assign$ ) then
  return Fail
else if Cost( $assign$ ) >  $o$ 
  return Fail
else if Complete( $assign$ )
  return  $assign$ 
end if
 $u \leftarrow$  next element of  $q$ 
 $newassign \leftarrow assign$  with  $u$  assigned to true
 $b1 \leftarrow$  call DPLL-RECUR( $c, newassign, o, b$ )
 $newassign \leftarrow assign$  with  $u$  assigned to false
 $b2 \leftarrow$  call DPLL-RECUR( $c, newassign, o, b$ )
if Cost( $b1$ ) < Cost( $b2$ ) then
  return  $b1$ 
else
  return  $b2$ 
end if

```

The elaboration step in basic DPLL is called unit-propagation. Unit-propagation examines the current assignment to determine if there are any constraints that have exactly one literal unassigned. If such constraints exist and exactly one assignment (i.e., true or false) for that literal satisfies the constraint, then DPLL makes that assignment immediately instead of through a later assumption. Our system augments this basic technique by introducing several more specialized forms of inference.

To understand the importance of elaboration, consider that all of the best available complete SAT-solvers are based on some version of DPLL (Moskewicz, Madigan et al. 2001; Een and Sorensson 2005). DPLL is so effective because its elaboration step eliminates the need to explore large numbers of unnecessary assumptions. It is more efficient to infer assignments directly rather than to make assumptions, because each assumption is equivalent to creating a new branch in DPLL’s abstract search tree. Elaboration also allows early detection of contradictions in the current assignment.

Despite its elaboration step, DPLL is unable to handle the large search spaces that occur when time is explicitly represented. The goal of our approach is to improve elaboration by using a combination of specialized inference routines. Previous work (Cassimatis, Bugjaska et al. 2007) has outlined the implementation of SAT solvers in Polyscheme. Following that approach, we implemented DPLL using Polyscheme’s focus of attention. One call to DPLL-RECUR is implemented by one focus of attention in Polyscheme. Logical worlds are used to manage DPLL assumptions. For each assumption, an alternative world is created in which the literal in question is either true or false. Once Polyscheme focuses on an assumption literal, it is elaborated by polling the opinions of several specialists. These specialists implement the specialized inference routines upon which our system relies. One of these specialists, the rule matcher, lazily instantiates grounded constraints that involve the current assumption. The assignment of a literal is given by Polyscheme’s final consensus on the corresponding proposition. This elaboration constitutes the main difference between our system and standard DPLL.

Our elaboration step, which is illustrated in Algorithm 3, loops over the literals that have been added to the queue because their assignments were modified by previous inference. Two procedures are performed on each of these literals. First, a rule matcher is used to lazily instantiate grounded constraints from relevant variable constraints provided in the input. Relevant constraints are those that contain a term that corresponds to the current literal in focus. These constraints are “fired” to propagate truth value from the antecedent terms to the consequent terms. Newly grounded literals, which may contain new objects, are introduced during this process. Any such literals are added to the assignment store and the queue.

The second procedure involves formulating an assignment for the current proposition based on suggestions from the various components of the system. For instance, the temporal constraint graph is queried here in the case that the proposition being examined describes a temporal relationship. Likewise, the identity constraint graph would be queried if the examined proposition was an identity relationship. In the extended system, this is the step at which the memory retrieval mechanism would be utilized. These opinions are combined with the old assignment of the proposition to produce a new assignment. If the new assignment differs from the old one, the literal is placed back on the queue.

Algorithm 3. ELABORATION($c, assign, q$):

```

while  $q$  is not empty do
   $l \leftarrow$  the next element in  $q$ 
   $ris \leftarrow$  call Match( $l, c, q$ )
   $delta \leftarrow \emptyset$ 
  for each  $ri$  in  $ris$  do
     $delta \leftarrow delta \cup$  call Propagate( $ri, assign$ )
  end for

```



```

rs ← the rule system's opinion on l
tc ← the temporal constraint graph's opinion on l
ic ← the identity constraint graph's opinion on l
mr ← the memory retrieval system's opinion on l
c ← call Combine(rs, tc, ic, mr)
if c ≠ l's assignment in assign then
    delta ← delta ∪ l
end if
q ← q ∪ delta
end while
return

```

Rule Matching Component

Lazy instantiation is efficient because it avoids the creation of constraints that do not need to be considered to produce an acceptable assignment. We accomplish lazy instantiation by treating constraints with open variables as templates that can be passed to a rule matcher. The rule matching component (RMC) attempts to bind the arguments of the last dequeued literal to variables in the constraint rules. A binding is valid if it allows all variables in the antecedent terms of a constraint to be bound to objects in the arguments of literals that are stored in the system's memory. All valid bindings are evaluated as constraints by propagating truth value from the antecedents to the consequents. A constraint is considered broken only if the grounded literals in its antecedent terms have been assigned to true and the propositions in its consequent terms have been assigned to false.

A simple example will illustrate the binding process. Let the literal currently being assigned be *Location(car1, road, t1)*. If *Location(?x, ?y, ?t1) ⇒ (10) Location(?x, ?y, ?t2) ∧ Meets(?t1, ?t2, E)* appears as a constraint, then the following fully grounded instance is created:

Location(car1, road, t1) ⇒ (10) Location(car1, road, tnew) ∧ Meets(t1, tnew, E)

In this case, *?x* binds to *car1*, *?y* binds to *road*, *?t1* binds to *t1*, and *?t2* binds to a new object, *tnew*. If *Location(car1, road, t1)* is assigned to true and *Location(car1, road, tnew)* is later assigned to false, then any models that contain that assignment will accrue a cost of 10.

Temporal Constraint Graph Component

As the number of objects in a problem instance increase, so do the number of literals and constraints. For instance, when time objects are introduced, it is often necessary to know how those times are ordered. If there are *n* times in the system, approximately $n^2/2$ literals are required to represent all of the values of a binary relation over those times. Usually, only a small portion of these literals provide useful information for solving the problem. Instead of eagerly encoding all temporal relations, we created a component that could be queried on demand to determine if a given relation holds according to the current knowledge of the system. This component represents relations in graphical form.

Using a graph enables the system to derive new entailed relations without storing them explicitly as propositions. All of the fundamental temporal relationships described by Allen can be represented in the following way. Whenever a literal that involves such a relationship is encountered, the temporal constraint graph (TCGC) decomposes the time object arguments into three parts: the start point, the midpoints, and the end point. These parts form the nodes of the graph. Edges are created between two nodes in the graph if their temporal relationship is known.

Every interval relationship can be derived from only two types of relationships on the parts of times: Before and Equals. A time object's start point is defined to be before its midpoints and its midpoints are before its end point. By creating edges between the parts of different time objects, it is possible to record relationships between the objects themselves. For instance, to encode *Meets(t1, t2, E)* in the graph, one would use the relationship: *Equals(end-t1, start-t2, E)*. To illustrate why the graph is an efficient way to store this information, consider the following example. If it is known that *Meets(t1, t2, E)* and *Meets(t2, t3, E)*, then the graph can be traversed to find *Before(t1, t3, E)*, among other relationships. Thus, these propositions are stored implicitly and do not need to be assigned unless they are present in a grounded constraint.

Identity Constraint Graph Component

The identity constraint graph component (ICGC) is similar to the TCGC, but it handles propositions about the identity of objects. This graph consists of nodes that represent objects and edges that represent either equality or inequality. By traversing the graph, it is possible to capture the transitivity of the identity relation. Although a rule could be used to generate the transitivity property, doing so has the potential to drastically increase the number of propositions over which DPLL must search.

Another important use for the ICGC is that it can detect inconsistencies in truth assignments to identity propositions. Consider that the following set of identity propositions is known: *Same(a, b)*, *Same(b, c)*. Then, the proposition, $\neg \text{Same}(a, c)$, is examined and assumed to be true. Clearly this is inconsistent, but without including a rule that defines the properties of identity, the system will continue to perform inference based on these facts until an explicit contradiction is encountered. Traversing the edges connecting *a*, *b*, and *c* in the graph will indicate that this scenario is contradictory. This information can be reported to the system as a whole during elaboration.

Memory Retrieval Component

Systems that explicitly represent time often also have to reason about change. However, in situations with imperfect knowledge, reasoning about change can be difficult. Consider the following information about the color of a car, assuming that *Location* is an attribute: *Color(car, red, t1)*, *Color(car, blue, t6)*, *Color(car, green, t11)*. In between *t1* and *t6* and *t6* and *t11*, it is consistent for the car to be

any color. However, in many scenarios, the car would remain *red* until it was painted *blue* and then *blue* until it was painted *green*. Although the world is dynamic, it also exhibits inertia. The principle of change minimization states that changes to particular objects are relatively infrequent and when changes do occur, they will be salient. It is worthwhile for a reasoning system to bet on such recurring patterns, because doing so significantly reduces the complexity of many problems.

We can frame the change minimization problem as a weighted SAT problem as follows. Given a set of attribute literals that involve the same first term, but whose second term changes over time, determine the minimum number of changes required to explain the data. To this end, constraints can be defined so that the least costly models will be those that exhibit the smallest possible number of attribute value changes. The memory retrieval component (MRC) is designed to accommodate this procedure by regulating which attribute values appear in grounded constraints. Only attribute values with some prior evidence in memory are considered. As an example, the car could have been *purple* at t_2 and *brown* at t_3 , but models that contain such literals are automatically excluded.

To control which values are considered, the MRC makes a copy of each literal that encodes an attribute with a native time index. This copy is modified to contain a time index that corresponds to an open-ended interval. Only when new attribute values are observed will that interval's endpoints be constrained. For instance, when the system elaborates $Color(car, red, t_1)$, a new literal $Color(car, red, t-car-red)$ will be introduced. Because it uses a content-addressable retrieval system, the MRC will henceforth report that the color of the car is likely to be red at every time point until new information is discovered. If the literal $Color(car, blue, t_6)$ is observed, then the literal $Color(car, blue, t-car-blue)$ will be introduced. Also, constraints will be added to limit the right side of the $t-car-red$ interval at t_6 and the left side of the $t-car-blue$ interval at t_1 .

Even with this optimization, change minimization is expensive if performed naively, since there are $\frac{(n-1)(n)}{2}$ possible transitions between n attribute values if nothing is known about when the values hold relative to each other. For instance, if a car is seen to be *red*, then *blue*, then *green*, the naïve formulation will consider such possibilities as a change from *green* to *red* even though this is clearly impossible. The change detection mechanism in the MRC utilizes the TCGC in conjunction with content-accessible memory to significantly reduce the number of impossible changes that are considered by the system. When an attribute literal containing an interval is elaborated, the only values that are adjacent in time to the current interval will be considered. Since the intervals do not have completely fixed endpoints, these neighboring times can be detected by looking at what times the current interval could possibly include. For instance, once it has been established that there is a change between t_1 and t_6 , the interval created around t_1 can no longer include t_{11} , so

the location t_1 will not appear as a possible previous state of the location at t_{11} .

Results

Our system was designed to improve the efficiency of applying SAT-solving to problems that involve an explicit representation of time. In order to accomplish this goal, we used Polyscheme to implement a DPLL-like algorithm with specialized forms of inference that permitted the creation of new objects. The ability to introduce new objects allowed us to use intervals to reduce the search space of these problems.

In order to test the level of improvement gained by the ability to add new objects independently of other techniques, we ran an evaluation with the memory retrieval component deactivated. The task we selected was optimal path planning through space. We represented this space as a graph. Although the particular problem we used did not model a changing environment, an explicit represent of time would be required if actions had side effects or if objects in the environment had changing states. For instance, a particular environment might contain walls that crumble over time. We asked the system to find the shortest valid path between two particular locations on the graph. Valid paths consisted of movements between adjacent locations that did not contain active obstacles. These types of problems are important to the object tracking and motion planning domains.

We initially compared our system against LazySAT because it is similar to our approach in that it also supports the lazy instantiation of constraints. Through experimentation, we determined what values of LazySAT's parameters enabled high performance on this task. LazySAT, however, is based on a randomized local search, which is inefficient for many structured domains. Therefore, we ran the same set of tests on MiniMaxSAT (Heras, Larrosa et al. 2007), which is one of the best DPLL-based systematic SAT-solvers. Because Markov logic is not complete and cannot report unsatisfiability, we were forced to select a configuration of space in which a valid path existed.

The evaluation problem involved a 9-location graph with one obstacle, which had to be circumvented. We were limited to 9 locations because the performance of the systems degraded on larger problems. To determine how well each system handled time, we created ten versions of the problem from 5 to 50 time points in increments of 5. For each condition and each system, we ran 10 trials and recorded the average runtime. These results are displayed in Figure 1. While our system was consistently better than LazySAT, MiniMaxSAT outperformed both until it had to contend with more than 30 time points. Our system required approximately constant time to solve the problem due to the fact that intervals make this task equivalent to the case of planning without explicit time points.

A second evaluation was conducted to show that the memory retrieval subsystem allowed change minimization

problems to be solved efficiently. We ran these tests on the system with and without the MRC activated. These problems consisted of a number of attribute value observations that involved the color of an object. For instance, suppose the following facts were given as input: *Color(car, red, t1)*, *Color(car, blue, t6)*, and *Color(car, green, t11)*. The system would then be tasked with finding the least costly model that explained this data, namely that the color changed from *red* to *blue* and then from *blue* to *green*.

The results from this evaluation are depicted in Figure 2. Not only does enabling the MRC permit the change minimization problems to be solved in less time than with the basic system, but it also increases the upper limit on problem size. Without the MRC, our system ran for over 50,000 seconds attempting to solve the 5 attribute value problem. These results demonstrate that the elimination of irrelevant constraints is a powerful technique for improving the performance of SAT-solvers.

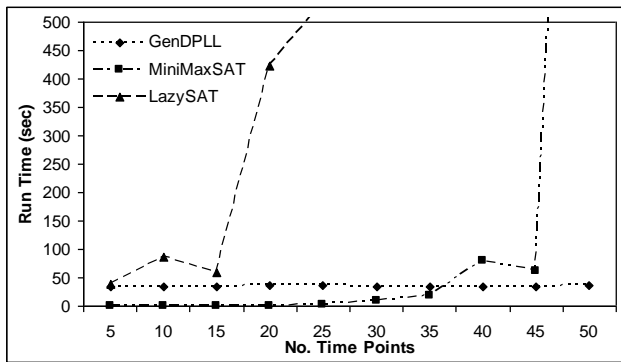


Figure 1: Path planning problem results

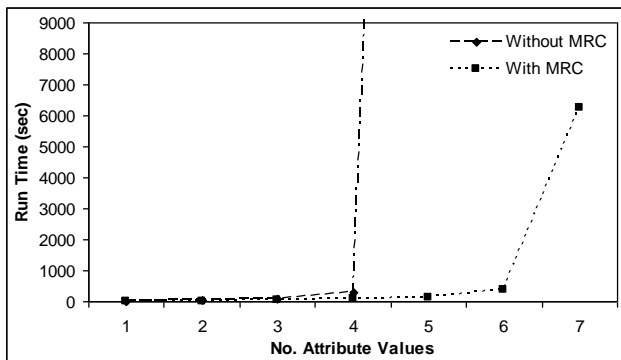


Figure 2: Change minimization results

Conclusion

Although SAT is in some cases an efficient approach to domain-general problem solving, it does not scale well to the large search spaces that result from tasks that require an explicit representation of time. Temporal intervals help to reduce the size of such problems, but can be used effectively only with SAT-solvers that permit the introduction of novel objects throughout inference. Hence,

we created a system that combines specialized inference techniques with a DPLL-like algorithm. This system was shown to outperform MiniMaxSAT and LazySAT in a series of evaluations involving a simple path planning domain.

When time is represented explicitly, it is also beneficial to incorporate common sense reasoning that exploits common patterns in real-world problem instances. Towards this end, a memory retrieval subsystem was developed that prevented the exploration of fruitless paths in the DPLL search tree under certain conditions. This technique was demonstrated to increase the efficiency of how our system solves the change minimization problem.

References

- Allen, J. (1981). Maintaining knowledge about temporal intervals. TR-86, Computer Science Department, University of Rochester, Rochester, NY.
- Cassimatis, N. L. (2002). Polyscheme: A Cognitive Architecture for Integrating Multiple Representation and Inference Schemes. Media Laboratory. Cambridge, MA, Massachusetts Institute of Technology.
- Cassimatis, N. L., M. Bugjaska, et al. (2007). An Architecture for Adaptive Algorithmic Hybrids. AAAI-07, Vancouver, BC.
- Davis, M., G. Logemann, et al. (1962). "A Machine Program for Theorem Proving." Communications of the ACM 5(7): 394-397.
- Een, N. and N. Sorensson (2005). MiniSat-A SAT solver with conflict-clause minimization. SAT 2005 Competition.
- Feldman, R. and M. C. Golumbic (1990). "Optimization algorithms for student scheduling via constraint satisfiability." Computer Journal 33: 356-364.
- Heras, F., J. Larrosa, et al. (2007). "MiniMaxSAT: a new weight Max-SAT solver." International Conference on Theory and Application of Satisfiability Testing: 41-55.
- Kautz, H. and B. Selman (1999). Unifying SAT-based and Graph-based Planning. IJCAI-99.
- Marques-Silva, J. P. and K. A. Sakallah (2000). "Boolean satisfiability in electronic design automation." Proc., IEEE/ACM Design Automation Conference (DAC '00).
- Moskewicz, M., C. Madigan, et al. (2001). Chaff: Engineering an Efficient SAT Solver 39th Design Automation Conference, Las Vegas.
- Mueller, E. T. (2004). "Event calculus reasoning through satisfiability." Journal of Logic and Computation 14(5): 703-730.
- Russell, S. and P. Norvig (1995). Artificial Intelligence: A Modern Approach. Prentice Hall.
- Shanahan, M. (1997). Solving the Frame Problem, a mathematical investigation of the common sense law of inertia. M.I.T. Press.
- Shanahan, M. and M. Witkowski (2004). "Event calculus planning through satisfiability." Journal of Logic and Computation 14(5): 731-745.
- Singla, P. and P. Domingos (2006). Memory-efficient inference in relational domains. Proceedings of the Twenty-First National Conference on Artificial Intelligence, Boston, MA.
- Smith, A. and A. Veneris (2005). "Fault diagnosis and logic debugging using boolean satisfiability." IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 24.

Discovering and Characterizing Hidden Variables

Soumi Ray and Tim Oates

Department of Computer Science and Electrical Engineering
University of Maryland Baltimore County
Baltimore, MD 21250

Abstract

Theoretical entities are aspects of the world that cannot be sensed directly but that nevertheless are causally relevant. Scientific inquiry has uncovered many such entities, such as black holes and dark matter. We claim that theoretical entities are important for the development of concepts within the lifetime of an individual, and present a novel neural network architecture that solves three problems related to theoretical entities: (1) discovering that they exist, (2) determining their number, and (3) computing their values. Experiments show the utility of the proposed approach using a discrete time dynamical system in which some of the state variables are hidden, and sensor data obtained from the camera of a mobile robot in which the sizes and locations of objects in the visual field are observed but their sizes and locations (distances) in the three-dimensional world are not.

Introduction

Humans, like robots, have limited sensory access to the physical world. Despite this fact, thousands of years of scientific inquiry have uncovered much hidden structure governing the behavior and appearance of the world, along the way proposing a vast array of entities that we cannot see, hear, taste, smell, or touch. When Gregor Mendel discovered genes in the middle of the 19th century, he couldn't experience them in the same way he could experience, say, the smell of a rose or the color red. However, genes have causal power that manifests itself in ways that can be sensed directly. For Mendel, one such manifestation was the color of the peas of the pea plants that he bred with one another. Whether a plant would have yellow or green peas could not be predicted accurately based solely on observable properties of the parent plants. The desire to explain this apparent non-determinism led Mendel to posit the existence of *a causally efficacious entity of the world that could not be sensed directly*, i.e., genes. Such entities are called *theoretical entities*.

Theoretical entities are of fundamental importance to the development of human knowledge. The history of science is replete with reliance on and corroboration of the existence of theoretical entities, like genes, atoms,

gravity, tectonic plates, germs, dark matter, electricity, and black holes. No one has ever seen a black hole, yet most physicists believe they exist because black holes accurately explain a wide array of observational data. Human knowledge would be limited indeed were we restricted to only represent and reason about things that we can see, hear, taste, smell, or touch.

This paper presents a novel neural network architecture for discovering hidden variables in time series data. The architecture is able to discover the existence of hidden variables, determine their number, and estimate their values. Empirical results are presented for a discrete time dynamical system and data gathered from a robots interactions with objects.

Background

McCallum (1996) did early work on hidden states in the context of reinforcement learning, where hidden states are typically ignored and traditional reinforcement learning methods are applied in fixed-memory agents. In other cases an agent's current percepts are augmented with history information. The problem in this situation is one of memory and storage. McCallum proposed a method called *instance-based state identification*, where raw data from previous experiences are stored directly. The simplest instance-based technique is the *nearest sequence memory* which is based on k-nearest neighbors. This technique, though simple, improved the performance of learning and took fewer training steps for learning. The main disadvantage of this technique is that, though it learns good policies quickly, it does not always learn the optimal policy.

Significant research in the recent past has focused on the problem of learning Bayesian Networks (BN) from data. Elidan *et al.* (2000) presents an algorithm for discovering hidden variables in Bayesian Networks by looking for cliques in network structures learned assuming all variables are observable. When a hidden variable is known to exist, they introduce it into the network and apply known BN learning algorithms. First, using the standard Bayesian model selection algorithm, a structure over the observed variables is learned. Then the structure is searched for sub-structures which they call semi-cliques. A hidden variable is then introduced to

break this clique and then learning is continued based on that new structure.

Similarly, work on planning under uncertainty using, for example, the Partially Observable Markov Decision Process (POMDP) framework assumes knowledge of the number of underlying hidden states (Kaelbling, Littman, & Cassandra, 1996). The agent whose world is characterized by the POMDP does not have access to the state that it actually occupies. Rather, the agent maintains a belief state, or probability distribution over states that it might be occupying. This belief state is Markovian, meaning that no additional information from the past would help increase the expected reward of the agent. Again, the goal of this work is not to discover the existence of a hidden state, but to behave optimally given knowledge of the existence of hidden state. More recently, Littman, Sutton, & Singh (2001) showed that dynamical systems can be represented using Predictive State Representations (PSRs), or multi-step, action-conditional predictions of future observations, and that every POMDP has an equivalent PSR. PSRs can look both at the past and summarize what happened and can also look to the future and predict what will happen. PSR is a vector of tests (Rivest & Schapire, 1994) which stores the predictions for a selected set of action-observation sequences. Holmes & Isbell (2006) showed that the unobserved or hidden states can be fully captured by a finite history based representation called a looping prediction suffix tree (PST). They focus on cases of POMDPs where the underlying transition and observation functions are deterministic.

Latent variables are important in the Psychology and Social Science research. Bollen (2002) described three definitions of latent variables: *local independence*, *expected value true score*, and *non-deterministic functions of observed variables* and introduced a new notion of latent variables called "*sample generalizations*". Latent variables can be defined *non-formally* or *formally*. Non-formally latent variables can be considered as *hypothetical variables* or unobserved variables as a data reduction device. Hypothetical variables are variables considered imaginary, i.e. not existing in the real world. Unobservable variables are impossible to be measured. The third non-formal definition of latent variables defines them as a data reduction device that can be used to describe a number of variables by a small number of factors.

One of the most common and popular formal definitions of latent variables is the *local independence* definition (Lord 1953, Lazarsfeld 1959, McDonald 1981, Bartholomew 1987, Hambleton et al. 1991). It means that the observed variables are associated with each other because of one or more latent variables. If the latent variables are known and are held constant then the observed variables become independent. This can be defined more formally:

$$P(Y_1, Y_2, \dots, Y_k | N) = P(Y_1 | N) P(Y_2 | N) \dots P(Y_k | N) \quad (1)$$

where Y_1, Y_2, \dots, Y_k are random observed variables and N is a vector of latent variables.

The next formal definition of latent variables defines a *true score*. The true score is calculated as the expected value of the observed variable for a particular object. Another definition of latent variables is that latent variables are non-deterministic functions of the observed variables, that is, they cannot be expressed as a function of the observed variables (Bentler 1982). It might be possible to predict a value of the latent variable but it is not possible to exactly predict the value of the latent variable based on the observed variables. The definition introduced by Bollen for latent variables is *sample realization*. He said that a latent variable is a random (or nonrandom) variable for which there is no sample realization for some observations, that is, there are no values for those observations. Observed variables contain sample realization while latent variables do not.

Some of the useful properties of latent variables were also discussed in Bollen's paper. A latent variable is denoted as a *posteriori* if it derived from a data analysis. On the other hand, *a priori* latent variables are hypothesized before the data analysis is done. Another property of latent variables can be understood by finding if they are affected by the observed variables or observed variables are the effects of the latent variables.

We are interested in finding hidden variables in time series data in a partially observable environment. We are not only interested in discovering hidden variables but also find the number of hidden variables in a given situation.

Method

We have designed a novel neural network architecture for the discovery and quantification of hidden variables. Our neural network architecture is comprised of two linked networks, the original network (O net) and the latent network (L net), as shown in figure 1. We call this network the *LO net*. History values of the observed process are input into each component network, and the output of the L net is also an input of the O net.

Consider the problem of predicting the value of a variable x at time $t+1$, given information up to time t . The current and previous two values are provided as inputs to both the original and the latent network and the next value is predicted. The input to the latent network is the current and previous two values at all times. The input to the original network is initially all three values, but with more learning the history values are dropped sequentially. This is done to give more responsibility to the learning of the latent network and the latent network can learn in such a way to output an approximation to the hidden variables. The network is trained using gradient descent backpropagation.

Since the latent network is not provided with any example outputs, the only way it learns is from the errors back-propagated to it. We want the latent network to learn the value of the hidden variable. The idea behind dropping the history inputs from the original network

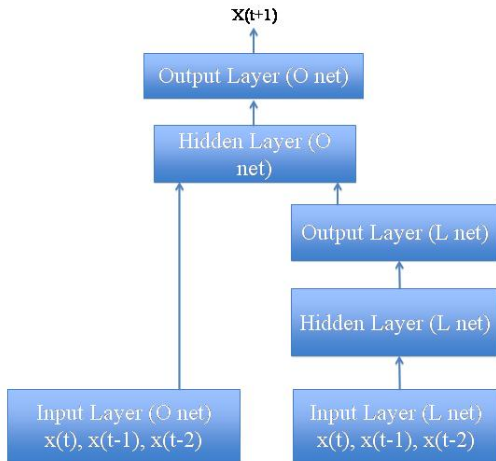


Figure 1: Our Network Architecture.

as learning progresses is to make the output from the latent network a crucial input to the original net. Hence the performance of the whole network will depend on what the latent network is learning and we expect the latent network to approximate the hidden variable. In our example we have taken a history of size two, i.e., the two previous observations. This method can work for smaller or larger history sizes. The history size will vary for different domains.

A theorem by Takens (Takens, 1981) states that for discrete-time deterministic dynamical systems of n variables, it is possible to exactly recover the topology of the system by treating each window of $2n$ consecutive values of just one variable as a state. This provides the basis for heuristic ideas in using history values to evaluate processes with hidden variables. The use of the neural network provides us the flexibility to mimic the transformation in Taken's theorem without worrying about its particular functional form. The neural network architecture thus provides a promising approach for estimating the true underlying system including hidden variables.

In our implementation the network has each of its layers' weights and biases initialized with the Nguyen-Widrow layer initialization method. The Nguyen-Widrow method generates initial weight and bias values for a layer so that the active regions of the layer's neurons are distributed approximately evenly over the input space. The values contain a degree of randomness, so they are not the same each time this function is called. The training function used to update the weight and bias values in the network is gradient descent with momentum and adaptive learning rate backpropagation. The parameter lr indicates the learning rate, similar to simple gradient descent. The parameter mc is the momentum constant that defines the amount of momentum. mc is set between 0 (no momentum) and values close to 1 (high momentum). A momentum

constant of 1 results in a network that is completely insensitive to the local gradient and, therefore, does not learn properly. The momentum constant (mc) used was 0.9. The learning rate (lr) we have chosen is 0.01. For each epoch, if performance decreases toward the goal, then the learning rate is increased by the factor $lr\text{-inc}$ (1.05). If performance increases by more than the factor max-perf-inc (1.04), the learning rate is adjusted by the factor $lr\text{-dec}$ (0.7) and the change, which increased the performance, is not made. A transfer function is used to calculate the i^{th} layer's output, given the layer's net input, during simulation and training. Backpropagation is used to calculate derivatives of performance ($perf$) with respect to the weight and bias variables X . The network's performance is measured according to the mean squared error. Each variable is adjusted according to gradient descent with momentum given in Eq 2,

$$dX = mc * dX_{\text{prev}} + lr * (1 - mc) * d\text{perf}/dX \quad (2)$$

where dX_{prev} is the previous change to the weight or bias. The transfer function used to calculate the hidden layer's output is the *tan-sigmoid* transfer function and the output layers use a *linear* transfer function.

Robot Data

Real world data was provided for this project by a surveyor SRV-1 Blackfin robot. The robot consists of a camera mounted on a pair of tank style treads that can be controlled remotely by a user interface on a laptop. The robot was placed in a fairly uniform environment (in this case the UMBC Department of Computer Science lobby) and driven by a human around a target. The targets consist of several brightly colored boxes, easily distinguishable from the surrounding environment by our image processing software. The surveyor would approach a target from different angles, keeping it in view the entire time for some trials, and for others occasionally facing different directions. Each frame transmitted from the surveyor's camera was recorded for later processing. The computation done on these frames consisted of counting the number of pixels that were present in a certain color range (giving us the surveyor's perception of the size of the box), and the centroid of the pixels of that color. Before each experiment, the color range was calibrated to avoid the surveyor mistaking other things for its target.

The absolute position of the robot in relation to its target was calculated by a camera hanging above the area in which the tests were being performed. The surveyor was tagged in its center with another unique color, and the camera was able to observe the position of the surveyor in relation to the box. This data was used to reconstruct the path of the robot, which was fitted across the data taken from the surveyor's camera in order to give us an approximation of the surveyor's position at each point.

The robot's vision system extracts the following information for a given box:

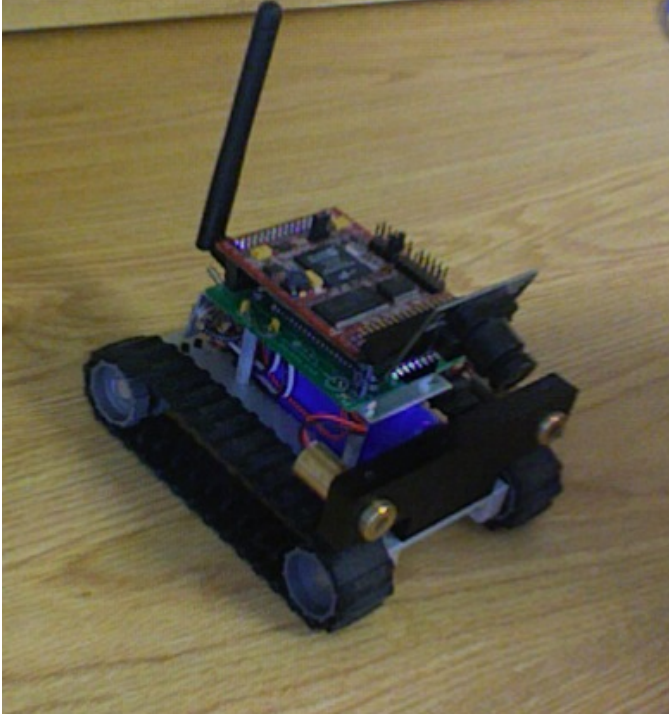


Figure 2: SRV-1 Blackfin Robot

s_i - the size of the object in the image plane.

x_i - the x coordinate of the object in the image plane

y_i - the y coordinate of the object in the image plane

Each object has an objective size s_0 and objective location (x_0, y_0, z_0) , relative to the origin of the coordinate frame.

In general, if the camera moves in the three-dimensional space with translational velocity $v = (v_x, v_y, v_z)$ and rotational velocity $\omega = (\omega_x, \omega_y, \omega_z)$, then the velocity of a point in the image plane can be expressed as follows:

$$\begin{aligned}\dot{x}_i &= t_x + r_x \\ \dot{y}_i &= t_y + r_y\end{aligned}$$

where,

$$\begin{aligned}t_x &= (-v_x + v_z x_i)/z_0 \\ t_y &= (-v_y + v_z y_i)/z_0 \\ r_x &= \omega_x x_i y_i - \omega_y (1 + x_i^2) + \omega_z y_i \\ r_y &= \omega_x (1 + y_i^2) - \omega_y x_i y_i - \omega_z x_i\end{aligned}$$

For our robot, the translational velocities v_x and v_y and rotational velocities w_x and w_z are physically constrained to be zero. So the equations for \dot{x}_i and \dot{y}_i are:

$$\dot{x}_i = v_z x_i / z_0 + w_y (1 + x_i^2) \quad (3)$$

$$\dot{y}_i = v_z y_i / z_0 + w_y x_i y_i \quad (4)$$

where v_z and w_y are constants.

The position of the image plane at each time step is then given by:

$$\begin{aligned}x_{t+1} &= x_t + \dot{x}_t \\ y_{t+1} &= y_t + \dot{y}_t\end{aligned}$$

Note that all the quantities required to predict the next value of x_{t+1} and y_{t+1} are observable except z_0 , the distance of the robot from the box.

The perceived size of an object s_i depends on the objective size s_0 and the distance z_0 of the object as follows:

$$s_i = s_0 / z_0^2 \quad (5)$$

The robot's perception of the size of the target thus changes with the distance from the target, though the target itself is of constant size. The quantities s_0 and z_0 are not observable, so s_i cannot be directly estimated. However, since s_0 is constant and our perspective projection is planar, we have a simpler situation where s_i changes only with z_0 .

Experiments

This section presents the results of using the LO net architecture to predict future output based on history with the robot data. A robot collects data by going back and forth looking at an object. It records the x and y coordinates of the box and the size of the box in its vision. The network is trained for 400 epochs since around that time the MSE converges to a very low value. The plots show the MSE of the last 150 epochs. Initially the MSE is very high (around a few hundred) but it drops rapidly to around 50 in just first 7 or 10 epochs. Figure 3 plots the MSE of variable x. The solid line shows the performance when the current value of x (x_t) is fed and the next value of x (x_{t+1}) is predicted, using only the original network. The dashed line shows the performance when the current and the previous two x values (x_t, x_{t-1}, x_{t-2}) are input to the original network and the next value of x x_{t+1} is predicted. The dash-dot line and the dotted lines show the performance with one and two latent networks respectively. Initially for the first 100 epochs the current and the previous two x values (x_t, x_{t-1}, x_{t-2}) are input to the original and latent networks. The output of the latent networks are also given as an input to the original network as shown in figure 1 and the next value of x (x_{t+1}) is predicted. In the next 100 epochs one history value x_{t-2} is dropped from the original network and training is continued. In the last 200 epochs the original network is input with only the current value of x (x_t) and the output from the latent network. All the four figures plot the MSE versus the number of epochs. In the first case there is only one network and the input is just the current value. The second case is where there is also just one network but there are three inputs, the current and two previous inputs. The third and the fourth case show the results of the LO net architecture. In the third case there is one latent network along with the original network. In the fourth there are

two latent networks. The x-axis plots the number of iterations and the y-axis plots the mean square error (MSE) in the following three figures.

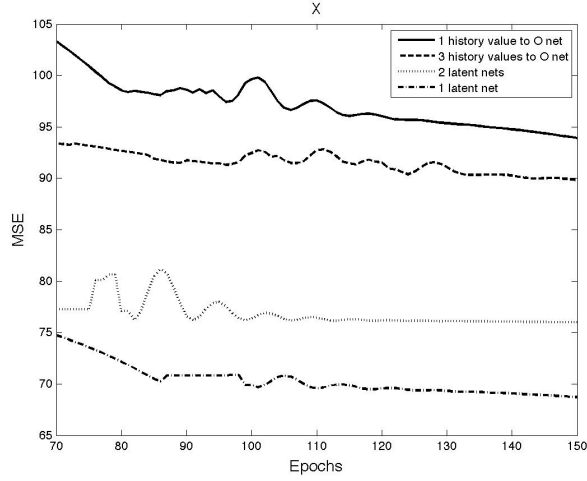


Figure 3: Performance curve for X.

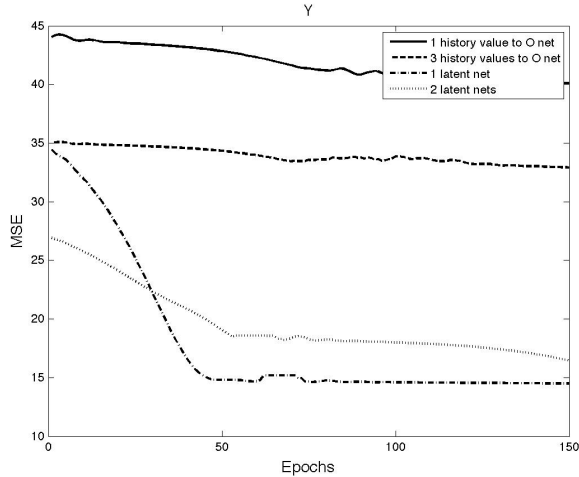


Figure 4: Performance curve for Y.

In figure 3 the performance of the network with three history values can be seen to be better than the performance with just the current value. The one LO net performs best in this case. It performs better than the two latent network architecture also. From equation 3 it is clear that there is one value which is unobservable for the prediction of x_{t+1} which is the distance of the robot from the box. While trying to predict the the next value of x with just the previous value of x one variable is hidden to x on which it is dependent. The output from the latent network in the LO net architecture provides input to the original input that improves its performance. The latent network posits the pres-

ence of a hidden variable. It approximately learns the value of the hidden variable. Initially three history values are provided as input to the original network but with more learning history values are dropped and so the input from the latent network becomes more important. We propose that the backpropagation algorithm updates the weights of the latent network in such a way so as to approximate the hidden variable. Similar results can be seen in the case of predicting y_{t+1} and s_{t+1} . The one latent network architecture improves the performance of learning in all the three cases. Adding a second latent network in these cases reduces the performance. There are two unobservable values for predicting the size of the box — the distance of the robot from the box and the actual size of the box. Since the actual size is constant the perceived size of the box changes only when the distance changes. So the latent network comes up with just one hidden variable in this case.

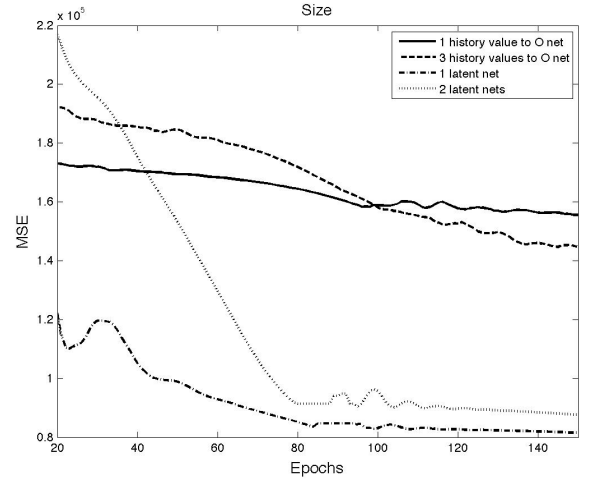


Figure 5: Performance curve for size.

The next experiments are performed on data where there are two boxes in the vision of the robot. The architecture with two latent networks performs best when predicting the size of the box as seen in figure 6. The size of a box depends on the actual size of the box and the distance of the box from the robot. When there are two boxes the actual size is no more constant. So when predicting the next value of the size perceived by the robot the two hidden variables are the size and the distance. Adding a third latent network again reduces the performance of learning.

From these results we conclude that the performance of prediction of the future values can be improved by using the LO net architecture. Not only does it estimate the existence of hidden variables but it also gives an estimate of the number of hidden variables. For example x_{t+1} and y_{t+1} depend only on one unobservable variable, so one latent network does a better job than two latent networks. In the two latent network case the extra input from the second latent net reduced the per-

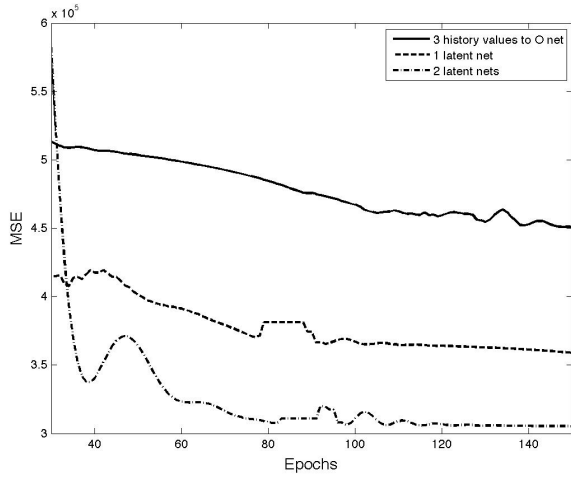


Figure 6: Performance curve for size with two boxes.

formance. While predicting the future values of s_{t+1} with boxes in the robot's view which depends on two unobservable variables, two latent nets did a better job than one. The network architecture was able to predict two hidden variables.

Figure 7 show the outputs of the latent networks from the three experiments with one latent network while trying to predict the next values of x , y and size with one box in the robot's vision. All the three latent networks try to approximate one variable which is hidden, the distance of the robot from the box. It can be seen that the outputs from the latent networks are somewhat correlated.

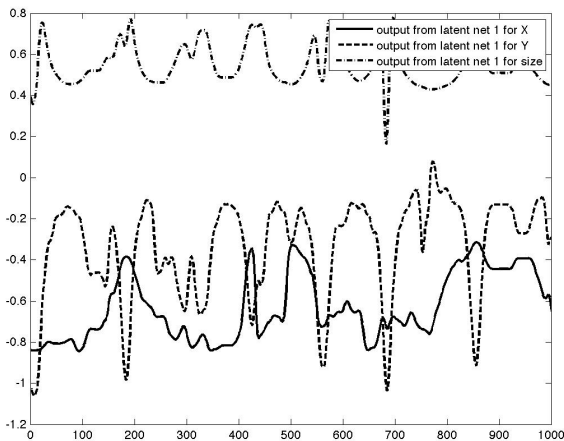


Figure 7: Comparison of the output values from the latent nets.

This neural network architecture can find the existence of hidden variables. The number of hidden variables can be found by iteratively adding latent networks

to the original network until adding a new latent network does not significantly help. Our next experiments will be on large domains to see how this method scales when the number of hidden variables increases.

Conclusion

We presented a novel neural network architecture that solves three problems related to theoretical entities: (1) discovering that they exist, (2) determining their number, and (3) computing their values. Experiments showed the utility of the proposed approach using a discrete time dynamical system in which some of the state variables are hidden, and sensor data obtained from the camera of a mobile robot in which the sizes and locations of objects in the visual field are observed but their sizes and locations (distances) in the three-dimensional world are not.

Acknowledgement

We would like to thank Max Morawski for running the experiments with the robot and providing the experimental data.

References

- Bollen, K. A. 2002. Latent variables in psychology and the social sciences. *Annual Review of Psychology* 53(1):605–634.
- Elidan, G.; Lotner, N.; Friedman, N.; and Koller, D. 2000. Discovering hidden variables: A structure-based approach. In *NIPS*, 479–485.
- Holmes, M. P., and Isbell, Jr., C. L. 2006. Looping suffix tree-based inference of partially observable hidden state. In *ICML '06: Proceedings of the 23rd international conference on Machine learning*, 409–416. New York, NY, USA: ACM.
- Kaelbling, L. P.; Littman, M. L.; and Cassandra, A. R. 1996. Planning and acting in partially observable stochastic domains. Technical Report CS-96-08.
- Littman, M.; Sutton, R.; and Singh, S. 2001. Predictive representations of state.
- McCallum, A. 1996. Hidden state and reinforcement learning with instance-based state identification.
- Rivest, R. L., and Schapire, R. E. 1994. Diversity-based inference of finite automata.
- Takens, F. 1981. Detecting strange attractors in turbulence. *Lecture Notes in Mathematics* 366–381.

Compression Progress, Pseudorandomness, & Hyperbolic Discounting

Moshe Looks

Google, Inc.

1600 Amphitheatre Pkwy, Mountain View, CA 94043

madscience@google.com

Abstract

General intelligence requires open-ended exploratory learning. The principle of compression progress proposes that agents should derive intrinsic reward from maximizing “interestingness”, the first derivative of compression progress over the agent’s history. Schmidhuber posits that such a drive can explain “essential aspects of ... curiosity, creativity, art, science, music, [and] jokes”, implying that such phenomena might be replicated in an artificial general intelligence programmed with such a drive. I pose two caveats: 1) as pointed out by Rayhawk, not everything that can be considered “interesting” according to this definition is interesting to humans; 2) because of (irrational) hyperbolic discounting of future rewards, humans have an additional preference for rewards that are structured to prevent premature satiation, often superseding intrinsic preferences for compression progress.

Consider an agent operating autonomously in a large and complex environment, absent frequent external reinforcement. Are there general principles the agent can use to understand its world and decide what to attend to? It has been observed going back to Leibniz that understanding is in many respects equivalent to compression.¹ To understand its world, a competent agent will thus attempt, perhaps implicitly, to compress its history through the present, consisting of its observations, actions, and external rewards (if any). Any regularities that we can find in our history through time t , $h(\leq t)$, may be encoded in a program p that generates the data $h(\leq t)$ as output by exploiting said regularities.

Schmidhuber has proposed the *principle of compression progress* (Sch09): long-lived autonomous agents that are computationally limited should be given intrinsic reward for increasing subjective “interestingness”, defined as the first derivative of compression progress (compressing $h(\leq t)$). Agents that are motivated by compression progress will seek out and focus on regions of their environment where such progress is expected. They will avoid both regions of the world which are entirely predictable (already highly compressed), and entirely unpredictable (incompressible and not expected to yield to compression progress).

A startling application of the principle of compression progress is to explain “essential aspects of subjective beauty, novelty, surprise, interestingness, attention, curiosity, creativity, art, science, music, jokes”, as attempted in (Sch09). The unifying theme in all of these activities, it is argued, is the active process of observing new data which provide for the discovery of novel patterns. These patterns explain the data as they unfold over time by allowing the observer to compress it more and more. This progress is explicit and formal in science and mathematics, while it may be implicit and even unconscious in art and music. To be clear, engaging in these activities often provides external rewards (fame and fortune) that are not addressed here; we consider only the intrinsic rewards from such pursuits.

Rayhawk (Ray09) criticizes this attempt with a gedankenexperiment. First, generate a (long) sequence of 2^n bits with a pseudorandom number generator (PRNG) using an unknown but accessible random seed, n bits long. Assuming that the PRNG is of high quality and our agent is computationally limited, such a sequence will require $\Theta(2^n)$ bits to store. Access the random seed, and use it to recode the original 2^n bits in $\Theta(n)$ space by storing just the seed and the constant-length PRNG code. This will lead to compression progress, which can be made as large as we would like by increasing n . Of course, such compression progress would be very uninteresting to most people!

The applicability of this procedure depends crucially on two factors: 1) how the complexity of compression programs is measured by the agent, namely the tradeoff between explanation size (in bits) and execution time (in elementary operation on bits); and 2) which sorts of compression programs may be found by the agent. Consider an agent that measures compression progress between times t and $t + 1$ by $C(p(t), h(\leq t + 1)) - C(p(t + 1), h(\leq t + 1))$ (see (Sch09) for details). Here $p(t)$ is the agent’s compression program at time t , and $C(p(t), h(\leq t + 1))$ is the cost to encode the agent’s history *through time* $t + 1$, with $p(t)$. If execution time is not accounted for in C (i.e. cost is simply the length of the compressor program), and p may be any primitive recursive program, the criticism disappears. This is because even without knowing the random seed, $O(n)$

¹Cf. (Bau04) for a modern formulation of this argument.

bits are sufficient to encode the sequence, since we can program a brute-force test of all possible seeds without incurring any complexity costs, while storing only a short prefix of the overall sequence. Thus, the seed is superfluous and provides no compression gain. If execution time has logarithmic cost relative to program size, as in the speed prior (Sch02), then learning the seed will provide us with at most a compression gain logarithmic in n . This is because testing all random seeds against a prefix of the the sequence takes $O(n2^n)$ time, so $C(p(t), h(\leq t+1))$ will be about $n + \log(n)$, while $C(p(t+1), h(\leq t+1))$ will be about n .

Thus, such pathological behavior will certainly not occur with a time-independent prior. Unfortunately, the compression progress principle is intended for precisely those computationally limited agents with time-dependent priors, that are too resource-constrained to brute-force random seeds. A reasonable alternative is to posit an *a priori* weighting over data that would assign zero utility to compression progress on such a sequence, and nonzero utility to compression of e.g. knowledge found in books, images of human faces, etc. This gives a principle of *weighted* compression progress that somewhat less elegant, but perhaps more practical.

A very different theory that also addresses the peculiar nature of intrinsic rewards in humans is hyperbolic discounting, based on long-standing results in operant conditioning (Her61). In standard utility theory, agents that discount future rewards against immediate rewards do so exponentially; an expected reward occurring t units of time in the future is assigned utility $r\gamma^t$ relative to its present utility of r , where γ is a constant between 0 and 1. The reason for the exponential form is that any other function leads to inconsistency of temporal preferences; what the agent prefers now will not be what it prefers in the future. However, considerable empirical evidence (Ain01) shows that humans and many animals discount future reward not exponentially, but hyperbolically, approximating $r(1+t)^{-1}$. Because of the hyperbolic curve's initial relative steepness, agents discounting according to this formula are in perpetual conflict with their future selves. Immediately available rewards can dominate decision-making to the detriment of cumulative reward, and agents are vulnerable to self-induced "premature satiation", a phenomenon that is nonexistent in exponential discounters (Ain01). While an exponential discounter may prefer a smaller sooner reward (when $\gamma < 1$), this preference will be entirely consistent over time; there will be no preference reversal as rewards become more imminent.

Hyperbolic discounting and the compression progress principle intersect when we consider activities that provide time-varying intrinsic rewards. *They conflict when rewards may be consumed at varying rates for varying amounts of total reward.* Consider an agent examining a complex painting or sculpture that is not instantaneously comprehensible, but must be understood sequentially through a series of attention-shifts to various parts. Schmidhuber (Sch09) asks: "Which sequences

of actions and resulting shifts of attention should he execute to maximize his pleasure?" and answers "According to our principle he should select one that maximizes the quickly learnable compressibility that is new, relative to his current knowledge and his (usually limited) way of incorporating / learning / compressing new data." But a hyperbolically discounting agent is incapable of selecting such a sequence voluntarily! Due to temporal skewing of action selection, a suboptimal sequence that provides more immediate rewards will be chosen instead. I posit that the experiences humans find most aesthetically rewarding are those with intrinsic reward, generated by weighted compression progress, *that are structured to naturally prevent premature satiation.*

In conclusion, I posit two major qualifications of the applicability of the principle of compression progress to humans. First, that the value of compression progress is weighted by the *a priori* importance of the data that are being compressed. This is most obvious in our interest in faces, interpersonal relations, etc. Even more abstract endeavors such as music (Mit06) and mathematics (LN01) are grounded in embodied experience, and only thus are such data worth compressing to begin with. Second, that experiences that intrinsically limit the "rate of consumption" of compression progress will be preferred to those requiring self-regulated consumption, even when less total reward is achievable by a rational agent in the former case than in the latter. AGI designers should bear these caveats in mind when constructing intrinsic motivations for their agents.

Acknowledgements Thanks to Steve Rayhawk and Jürgen Schmidhuber for helpful discussion.

References

- G. Ainslie. *Breakdown of Will*. Cambridge University Press, 2001.
- E. B. Baum. *What is Thought?* MIT Press, 2004.
- R. Herrnstein. Relative and absolute strength of response as a function of frequency of reinforcement. *Journal of the Experimental Analysis of Behavior*, 1961.
- G. Lakoff and R. Núñez. *Where Mathematics Comes From: How the Embodied Mind Brings Mathematics into Being*. Basic Books, 2001.
- S. J. Mithen. *The Singing Neanderthals: The Origins of Music, Language, Mind, and Body*. Harvard University Press, 2006.
- S. Rayhawk. Personal communication, 2009.
- J. Schmidhuber. The speed prior: a new simplicity measure yielding near-optimal computable predictions. In *Conference on Computational Learning Theory*, 2002.
- J. Schmidhuber. Driven by compression progress. In G. Pezzulo, M. V. Butz, O. Sigaud, and G. Baldassarre, editors, *Anticipatory Behavior in Adaptive Learning Systems*. Springer, 2009.

A minimum relative entropy principle for AGI

Antoine van de Ven* and Ben A.M. Schouten

Fontys University of Applied Sciences
Postbus 347, 5600 AH Eindhoven, The Netherlands
*Antoine.vandeVen@fontys.nl

Abstract

In this paper the principle of minimum relative entropy (PMRE) is proposed as a fundamental principle and idea that can be used in the field of AGI. It is shown to have a very strong mathematical foundation, that it is even more fundamental than Bayes rule or MaxEnt alone and that it can be related to neuroscience. Hierarchical structures, hierarchies in timescales and learning and generating sequences of sequences are some of the aspects that Friston (Fri09) described by using his free-energy principle. These are aspects of cognitive architectures that are in agreement with the foundations of hierarchical memory prediction frameworks (GH09). The PMRE is very similar and often equivalent to Friston's free-energy principle (Fri09), however for actions and the definitions of surprise there is a difference. It is proposed to use relative entropy as the standard definition of surprise. Experiments have shown that this is currently the best indicator of human surprise (IB09). The learning rate or interestingness can be defined as the rate of decrease of relative entropy, so curiosity can then be implemented as looking for situations with the highest learning rate.

Introduction

Just like physics wants to find the underlying laws of nature, it would be nice to find underlying principles for intelligence, inference, surprise and so on. A lot of progress has been made and many principles have been proposed. Depending on what principles or foundations are used, it is possible to come to different theories or implementations of intelligent agents. A good example is AIXI (Hut04) which combines Decision Theory with Solomonoff's universal induction (which combines principles from Ockham, Epicurus, Bayes and Turing). It uses compression and Kolmogorov complexity, but unfortunately this makes it uncomputable in this form. The ability to compress data well has been linked to intelligence and compression progress has been proposed as a simple algorithmic principle for discovery, curiosity and more. While this has a very strong and solid mathematical foundation, the problem is that it is often very hard or even impossible to compute. Often it is also assumed that the agent stores all data of all

sensory observations forever. It seems unlikely that the human brain works like that.

In (vdV09) the principle of minimum relative entropy (PMRE) was proposed to be used in developmental robotics. In this paper we want to propose it as a fundamental principle and idea for use in the field of AGI. We compare it with other principles, relate it to cognitive architectures and show that it can be used to model curiosity. It can be shown that it has a very solid and strong mathematical foundation because it can be derived from three simple axioms (Gif08). The most important assumption and axiom is the principle of minimal updating: beliefs should be updated only to the extent required by the new information. This is incorporated by a locality axiom. The other two axioms are only used to require coordinate invariance and consistency for independent subsystems. By eliminative induction this singles out the logarithmic relative entropy as the formula to minimize. This way the Kullback-Leibler divergence (KLD) (KL51) has been derived as the only correct and unique divergence to minimize. Other forms of divergences and relative entropies in the literature are excluded. It can be shown (Gif08) to be able to do everything orthodox Bayesian inference (which allows arbitrary priors) and MaxEnt (which allows arbitrary constraints) can do, but it can also process both forms simultaneously, which Bayes and MaxEnt cannot do alone. This has only been shown recently and is not well known yet. The current version of the most used textbook on Artificial Intelligence (RN02) doesn't even include the words relative entropy or Kullback-Leibler divergence yet.

Free-energy

While in our approach the PMRE with the KLD is the most fundamental, in other approaches exact Bayesian inference is often taken as most fundamental, and the KLD is then used to do approximate inference. The variational Bayes method is an example of this. It tries to find an approximate distribution of a true posterior distribution by minimizing the KLD between the approximate distribution and the true posterior distribution. Sometimes a free-energy formulation is used which yields the same solution when minimized, but which can

make the calculations easier. In fact the free-energy formulation is the same as the KLD with an extra term (Shannon surprise) that doesn't depend on the approximate distribution, so it doesn't influence the search for the best approximate distribution. In the field of neuroscience, Friston (Fri09) has proposed the minimum free-energy principle as a fundamental principle that could explain a lot about how the brain functions. For perception it is equal to minimizing the KLD, so it is equivalent to the PMRE in that respect. Friston showed that many properties and functions of the brain can be explained by using the free-energy principle, such as the hierarchical structure of the brain, a hierarchy of timescales in the brain and how it could learn and generate sequences of sequences. This is in agreement with the memory prediction framework (GH09). Note that this not only relates these principles to the brain, but that it can also guide the design and choice of cognitive architectures for artificial general intelligence.

Currently the brain is the only working proof that general intelligence is possible, so these principles and results could help and guide biologically inspired AGI. These results seem to confirm the foundations of biologically inspired frameworks which use hierarchical structures, spatio-temporal pattern recognition and the learning and generating of sequences of sequences.

Biologically plausible

The fact that the PMRE only does minimal updating of the beliefs makes it more biologically plausible than some other theories. For example AIXI (Hut04) isn't based on minimal updating, because it uses global compression including all historical data. Brains don't seem to work that way. When observing and learning there are physical changes in the brain to incorporate and encode the new information and new beliefs. Such physical changes are costly for an organism and should be avoided as much as possible, because of limited energy and limited resources. The PMRE avoids this by doing only minimal updating of the beliefs. It is related to compression because in this way it stores new information and beliefs in an efficient way.

A new definition of surprise

Besides the theoretical arguments we can also refer to experiments. Itti and Baldi (IB09) proposed a definition of Bayesian Surprise that is equal to the KLD between the prior and posterior beliefs of the observer. This again is the same formula as used by the PMRE. In experiments they showed that by calculating this they could predict with high precision where humans would look. This formula and definition was found to be more accurate than all other models they compared it with, like Shannon entropy, saliency and other measures. In their derivation Itti and Baldi picked the KLD as the best way to define Bayesian Surprise by referring to the work of Kullback. While we agree on this definition, it would also have been possible to pick another diver-

gence as a measure, because the KLD is just one out of a broader class of divergences called f-divergences. The benefit of the derivation of the PMRE is that it uniquely selects the KLD as the only consistent measure that can be used. So in this way the PMRE helps to select and confirm this definition of surprise.

Relative entropy and curiosity

Relative entropy can also be used to implement curiosity and exploration. In (SHS95) it was used for reinforcement driven information acquisition, but it can also be implemented in different ways. The rate in which the relative entropy decreases can be seen as the learning rate. Curiosity can then be implemented as looking for and exploring the situations with highest learning rate (interestingness). This can be compared with implementations of curiosity which use decrease of prediction errors or compression progress (Sch09).

References

- Karl Friston. The free-energy principle: a rough guide to the brain? *Trends in Cognitive Sciences*, 13(7):293–301, 2009.
- Dileep George and Jeff Hawkins. Towards a mathematical theory of cortical micro-circuits. *PLoS Comput Biol*, 5(10):e1000532, oct 2009.
- Adom Giffin. *Maximum Entropy: The Universal Method for Inference*. PhD thesis, Massey U., Albany, 2008.
- Marcus Hutter. *Universal Artificial Intelligence: Sequential Decisions Based On Algorithmic Probability*. Springer, 1 edition, November 2004.
- Laurent Itti and Pierre Baldi. Bayesian surprise attracts human attention. *Vision Research*, 49(10):1295–1306, June 2009.
- S. Kullback and R. A. Leibler. On information and sufficiency. *The Annals of Mathematical Statistics*, 22(1):79–86, March 1951.
- Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 2nd edition, December 2002.
- Jürgen Schmidhuber. Driven by compression progress: A simple principle explains essential aspects of subjective beauty, novelty, surprise, interestingness, attention, curiosity, creativity, art, science, music, jokes. In *Anticipatory Behavior in Adaptive Learning Systems: From Psychological Theories to Artificial Cognitive Systems*, pages 48–76. Springer-Verlag, 2009.
- Jan Storck, Sepp Hochreiter, and Jürgen Schmidhuber. Reinforcement driven information acquisition in Non-Deterministic environments. In *Proc. International Conference on Artificial Neural Networks (ICANN95)*, 1995.
- Antoine van de Ven. A minimum relative entropy principle for the brain. In *Proceedings of the Ninth International Conference on Epigenetic Robotics*. Lund University Cognitive Studies, 145, 2009.

Towards Automated Code Generation for Autonomous Mobile Robots

D. Kerr

Intelligent Systems
Research Centre
University of Ulster
Londonderry, BT48 7JL, UK

U. Nehmzow

Intelligent Systems
Research Centre
University of Ulster
Londonderry, BT48 7JL, UK

S.A. Billings

Department of Automatic Control
and Systems Engineering
University of Sheffield
Sheffield, S1 3JD, UK

Abstract

With the expected growth in mobile robotics the demand for expertise to develop robot control code will also increase. As end-users cannot be expected to develop this control code themselves, a more elegant solution would be to allow the end-users to *teach* the robot by demonstrating the task.

In this paper we show how route learning tasks may be “translated” directly into robot control code simply by observing the task. We show how automated code generation may be facilitated through system identification — which algorithmically and automatically transfers human behaviour into control code, using transparent mathematical functions. We provide two route learning examples where a mobile robot automatically obtains control code simply by observing human behaviour, identifying it using system identification, and copying the behaviour.

Introduction

Motivation

Mobile robotics will play an ever more important role in the future. We expect one growth area to be service robotics, especially home care robots for the elderly and infirm. Other important growth areas will be entertainment robotics and games, as well as security applications.

All of these applications require some high-level, sophisticated programming, but the bulk of the programming work will be “standard” components of robot control, that will consume a lot of programmer resources — resources that could be better used.

In this paper we show that it is possible for trajectory learning to obtain robot control code *automatically*, through system identification (Akanyeti et al., 2007): Control code was obtained by observing a human demonstrator following the desired route, and by translating his behaviour *directly* into code, *without programming*.

Learning by demonstration is by now a widely used technique in robotics (see, for instance, (Demiris, 2009) and (Demiris and Dearden, 2005)). In terms of application (route learning), (Coates et al., 2008) are perhaps the most interesting here to mention: an expert was

used to control a model helicopter, the desired optimal behaviour was obtained and modelled from a number sub-optimal demonstrations performed by the expert. This model was then used to control the helicopter. In contrast to our work, (Coates et al., 2008) use a specialist to provide the demonstration, and their control model incorporates *a priori* knowledge such as that the helicopter has to remain stationary for certain manoeuvres. In our experiments the trainer is not an expert in the task, and only needs the ability to demonstrate the behaviour to the robot.

The experiments in this paper develop our approach further, their purpose is to show that even more complex behaviour of an agent — route learning in this case — can be “translated” directly into robot control code, namely by observing it, identifying it, using system identification, and using the identified model of the observed behaviour control the robot (Figure 1).

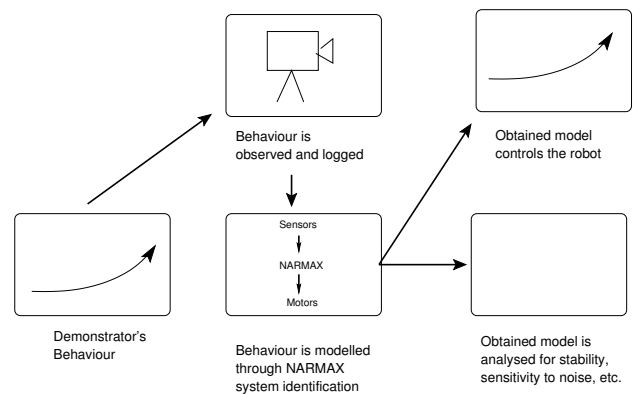


Figure 1: THE “BEHAVIOUR COPIER”: A BEHAVIOUR IS OBSERVED AND SUBSEQUENTLY IDENTIFIED USING SYSTEM IDENTIFICATION. THE OBTAINED MODEL IS THEN USED TO CONTROL THE ROBOT DIRECTLY, NO HUMAN INTERVENTION IS REQUIRED AT ANY POINT (OTHER THAN THAT THE HUMAN DEMONSTRATES THE DESIRED BEHAVIOUR TO THE ROBOT).

In essence the experiments reported here form a “behaviour copier”, which produces a canonical “carbon copy” of an observed behaviour that can be used to control an autonomous mobile robot.

Approach

We have used a NARMAX approach (Billings and Chen, 1998) to obtain the models we need for automated programming of a robot, because

- The Narmax model itself provides the executable code straight away,
- The model is analysable, and gives us valuable information regarding
 - How the robot achieves the task,
 - Whether the model is stable or not,
 - How the model will behave under certain operating conditions, and
 - How sensitive the model is to certain inputs, i.e. how “important” certain input are.

NARMAX system identification The NARMAX modelling approach is a parameter estimation methodology for identifying both the important model terms and the parameters of unknown non-linear dynamic systems. For multiple input, single output noiseless systems this model takes the form given in equation 1:

$$y(n) = f[\vec{u}(n - N_u)^l, y(n - N_y)],$$

$$\forall \quad N_u = 0 \dots N_u^{max},$$

$$l = 1 \dots l_{max}, N_y = 1 \dots N_y^{max}. \quad (1)$$

where $y(n)$ and $\vec{u}(n)$ are the sampled output and input signals at time n respectively, N_y and N_u are the regression orders of the output and input respectively. The input vector \vec{u} is d -dimensional, the output y is a scalar. $f()$ is a non-linear function and it is typically taken to be a polynomial or wavelet multi-resolution expansion of the arguments. The degree l_{max} of the polynomial is the highest sum of powers in any of its terms.

The NARMAX methodology breaks the modelling problem into the following steps: i) Structure detection (i.e. determining the form of the non-linear polynomial), ii) parameter estimation (i.e. obtaining the model coefficients), iii) model validation, iv) prediction, and v) analysis. A detailed description of how these steps are done is presented in (Billings and Chen, 1998; Korenberg et al., 1988; Billings and Voon, 1986).

The calculation of the NARMAX model parameters is an iterative process. Each iteration involves three steps: i) estimation of model parameters, ii) model validation and iii) removal of non-contributing terms.

Using System Identification to Obtain Robot-Executable Narmax models It is difficult for a programmer to teach a particular task to a robot as humans and robots perceive and act in the world differently; humans and robots have different sensor and actuator modalities (Alissandrakis et al., 2005). We have adopted a similar approach to (Nehmzow et al., 2007) where the mobile robot’s trajectory of the desired behaviour is used as a suitable communication channel between the human and the robot.

In this paper we present an approach to show how route learning can be translated directly into control code using system identification. The trajectory of the human is used as reference and this is translated algorithmically and automatically into robot control code. An outline this method is illustrated in Figure 2.

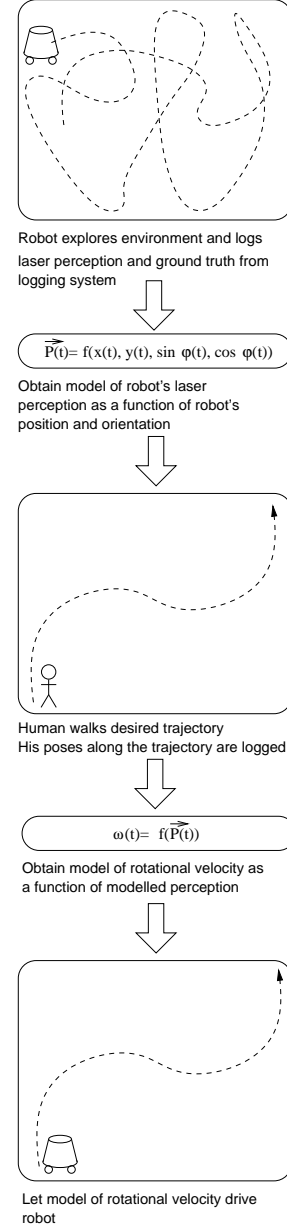


Figure 2: SYSTEM IDENTIFICATION PROCESS USED TO OBTAIN ROBOT-EXECUTABLE NARMAX MODELS

1. *Obtaining the sensorgraph:* The robot explores the environment and obtains a detailed log of sensor perceptions throughout the working environment. This detailed log, *the sensorgraph*, contains information such as laser readings and the robot’s pose $\langle x, y, \varphi \rangle$, i.e. its position $\langle x, y \rangle$ and heading φ within the environ-

ment.

2. *Obtain environment models:* The system identification method is used to obtain a number of polynomial models that model the robot's laser perception as a function of the robot's pose within the environment.

These models allow us to estimate the robot's laser perceptions along new novel trajectories that have been demonstrated by a human in the next stage.

3. *Obtaining a human-demonstrated trajectory:* The human user demonstrates the desired trajectory by performing the task in the target environment. During this demonstration period the demonstrator's pose is continuously observed and logged by a tracking system. These poses are then used to compute the translational and rotational velocities of the human by using consecutive poses along the trajectory.

4. *Obtain the final, environment-model-based, human demonstrated controller:* The final controller is obtained by using the system identification technique to obtain a sensor-based controller. The human demonstrator's location is used with the environment models (from stage 2) to obtain the robot's laser perception at that position. The modelled laser perceptions are then used as inputs to the system identification process with the computed velocities of the human demonstrator (from stage 3) as outputs.

5. *Robot copies behaviour:* The controller (from stage 4) can then be used to drive the robot along the human-demonstrated trajectory within the target environment, copying the behaviour of the human.

Experiments

The experiments in this paper were carried out in the robotics arena of the Intelligent Systems Research Centre in the University of Ulster. The robotics arena measures 100 m^2 and is equipped with a *Vicon* motion tracking system that delivers highly accurate position data (x, y, z) for targets using reflective markers and high speed high resolution cameras. In the experiments presented here we use the Metralabs *SCITOS G5* autonomous robot *Swilly*, shown in Figure 3.



Figure 3: *Swilly*, THE METRALABS SCITOS G5 MOBILE ROBOT USED IN THE EXPERIMENTS

The robot is equipped with 24 sonar sensors distributed around its circumference, and a *SICK* laser range finder, which scans the front of the robot $([0^\circ, 270^\circ])$ with a radial resolution of 0.5° . In our experiments the laser range finder was configured to scan the front semi-circle of the robot in the range $([0^\circ, 180^\circ])$.

Experimental Setup The robotics arena is configured with artificial walls to consist of a working environment measuring $4\text{ m} \times 3\text{ m}$, as illustrated in Figure 4.



Figure 4: OVERHEAD IMAGE OF THE ROBOT ARENA SETUP WITH THE ROBOT VISIBLE IN THE LOWER LEFT OF THE TEST AREA

The robot explores the test area using a random walk obstacle avoidance behaviour whilst simultaneously logging its laser perceptions and the robot's actual x, y, θ positions, obtained from the *Vicon* motion tracking system.

We ensure that adequate data to model the environment has been logged by computing histograms for the robot's actual position using the *Vicon* tracking system along the x -axis and y -axis. It is equally important to consider the robot's heading whilst exploring the environment as the modelling process needs to consider all possible orientations of the robot. Thus, we also construct a histogram of the robot's headings where the robot has logged sensor data. By obtaining an almost uniform distribution with the histograms we can ensure that adequate data has been logged.

The laser data is then median-filtered over 30° segments. So rather than 360 laser readings we have six median filtered segments that are used as input to the first modelling process.

Using the Narmax system identification method we obtain a number of polynomial models that model the robot's laser perception as a function of the robot's $\langle x, y \rangle$ position and heading φ (here we use $\sin(\varphi)$ and $\cos(\varphi)$) to form the function

$$\vec{P}(t) = f(x(t), y(t), \sin \varphi(t), \cos \varphi(t)) \quad (2)$$

The Narmax method used in this work has multiple inputs and a single output. Thus, we require at least one model per laser segment (6 in this case). When computing this model we need to consider all the possible (x, y) positional locations the robot may visit,

as well as the robots orientation φ at these positions. The dimensionality of this space is very high, and in order to manage the task of constructing a model we have restricted the number of laser models to only 4 models, modelling 90° heading segments, thus covering the entire 360° range of possible robot headings with 4 models. Put differently, we constructed four models, one for each 30° laser segment, of the form $P_k = f < x, y > \forall \varphi = k$, where P_k is a model of the predicted laser reading when the robot assumes a heading φ of k degrees.

The accuracy of the obtained environment models have been assessed by driving the robot along a novel trajectory within the test environment (see Figure 5) and logging the real laser perceptions along with the robot's position.

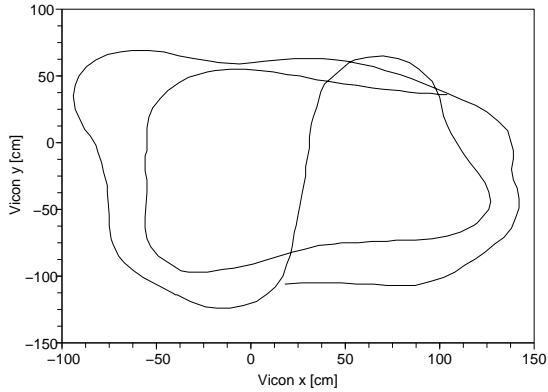


Figure 5: NOVEL TRAJECTORY DRIVEN BY THE ROBOT IN THE TEST AREA DURING WHICH REAL LASER DATA WAS LOGGED AND COMPARED WITH MODEL PREDICTED LASER DATA

The robot's position was used as input into the obtained environment models and the logged laser reading and modelled laser reading compared. In Figure 6 we have plotted the real and modelled laser values for laser segment 1, and plotted the absolute error between the values. The standard error for all the lasers is shown in Table 1.

Table 1: ABSOLUTE MEAN ERROR AND STANDARD ERROR IN CENTIMETRES FOR ALL LASERS OVER VALIDATION TRAJECTORY (FIGURE 5)

Laser	Mean and Standard error [cm]
\tilde{L}_1	14.7 ± 1.2
\tilde{L}_2	16.9 ± 1.0
\tilde{L}_3	16.9 ± 1.1
\tilde{L}_4	19.9 ± 1.6
\tilde{L}_5	16.9 ± 1.5
\tilde{L}_6	14.7 ± 1.1

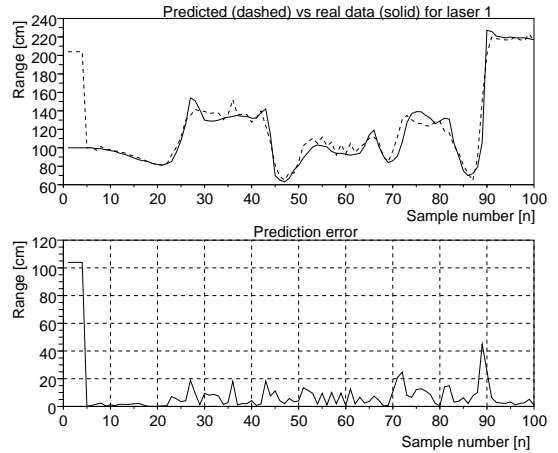


Figure 6: LOGGED TRACE FROM LASER 1 COMPARED WITH MODELLED TRACE FROM LASER 1 WHILST BEING DRIVEN ALONG THE TRAJECTORY IN FIGURE 5 AND ABSOLUTE ERROR BETWEEN LOGGED AND MODELLED LASERS.

Having obtained environment models with satisfactory accuracy we conduct experiments where a human follows two different trajectories, *S*-shaped and *U*-shaped.

Experiment 1 - *S*-shaped trajectory

The human demonstrated to the robot how to move within the test area in a *S*-shaped trajectory. The demonstrator started in the lower left side of the environment and walked in a *S*-shape finishing in the upper right side. The demonstrator's x and y position was again obtained from the *Vicon* system and logged every 250ms as the human demonstrator moved along the desired path 5-times in total. The logged trajectories are shown in Figure 7.

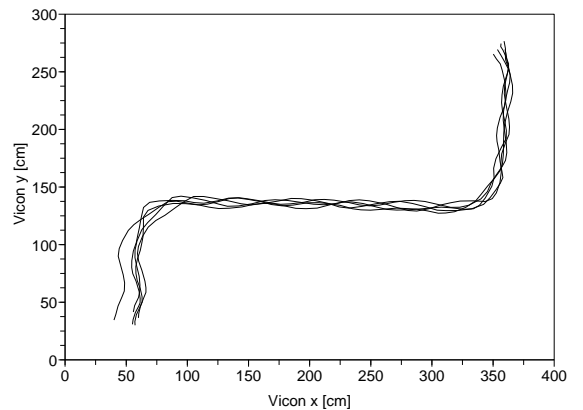


Figure 7: FIVE TRAJECTORIES OF THE DESIRED *S*-SHAPE BEHAVIOUR DEMONSTRATED BY THE HUMAN IN THE TEST ENVIRONMENT

The human demonstrator's $< x, y >$ positions are then used to compute the translational and rotational

velocities of the human along the trajectory by using consecutive $\langle x, y \rangle$ samples. Next, we obtain a series of expected robot perceptions along this trajectory of logged $\langle x, y \rangle$ positions using the environment models.

The final controller is obtained by using the Narmax system identification technique to obtain an environment-model sensor-based controller. The computed human demonstrator's rotational velocity is used with the series of expected robot perceptions forming the function

$$\omega(t) = f(\vec{P}(t)). \quad (3)$$

The modelled laser perceptions are used as inputs to the Narmax system with the computed rotational velocities of the human demonstrator as outputs. We used parameters with an input regression of 10 and polynomial degree 2 to obtain the following final Narmax model that had 20 terms, shown in equation 4.

$$\begin{aligned} \omega(t) = & -0.311 \\ & +0.001267 * u(n, 1) \\ & -0.007369 * u(n, 2) \\ & -0.001245 * u(n, 3) \\ & +0.00374 * u(n, 4) \\ & +0.00787 * u(n, 5) \\ & +0.00384 * u(n, 6) \\ & -0.0000078 * u(n, 1)^2 \\ & -0.000014 * u(n, 2)^2 \\ & -0.000011 * u(n, 3)^2 \\ & -0.0000053 * u(n, 4)^2 \\ & -0.0000034 * u(n, 6)^2 \\ & +0.000038 * u(n, 1) * u(n, 2) \\ & -0.000034 * u(n, 1) * u(n, 5) \\ & -0.0000059 * u(n, 1) * u(n, 6) \\ & -0.00000399 * u(n, 2) * u(n, 5) \\ & +0.0000163 * u(n, 3) * u(n, 4) \\ & +0.0000129 * u(n, 3) * u(n, 6) \\ & -0.00000166 * u(n, 4) * u(n, 5) \\ & -0.0000178 * u(n, 5) * u(n, 6) \end{aligned} \quad (4)$$

Results In the final stage of the experiment the obtained final controller is used to drive the robot whilst the robot's actual positions during this stage of the experiment are logged as illustrated in Figure 8.

A visual inspection of Figure 8 shows that, although not perfect, the obtained controller produces a behaviour that resembles the humans demonstrator's initial trajectory.

Experiment 2 - U-shaped trajectory

The human demonstrated to the robot how to move within the test area in a U-shaped trajectory. The

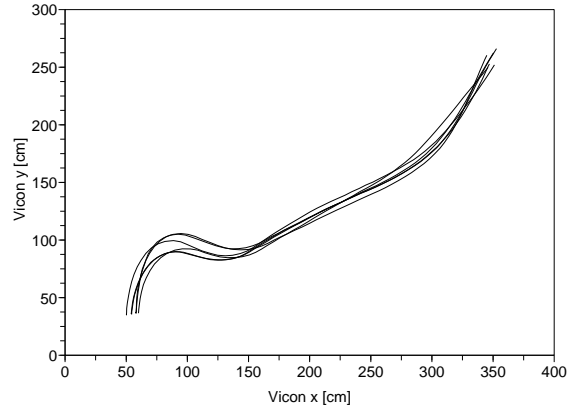


Figure 8: SIX TRAJECTORIES OF THE ROBOT UNDER CONTROL OF THE *S*-SHAPE SENSOR BASED CONTROLLER IN THE TEST ENVIRONMENT (COMPARE WITH FIGURE 7).

demonstrator started in the lower left side of the environment and walked in a U-shape finishing in the upper left side. The demonstrator's x and y position was again obtained from the Vicon system and logged every 250ms as the human demonstrator moved along the desired path 5-times in total. The logged trajectories are shown in Figure 9.

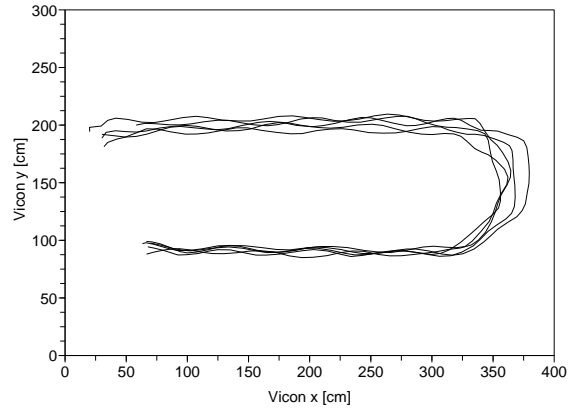


Figure 9: FIVE TRAJECTORIES OF THE DESIRED *U*-SHAPE BEHAVIOUR DEMONSTRATED BY THE HUMAN IN THE TEST ENVIRONMENT.

The human demonstrator's $\langle x, y \rangle$ positions are then used to compute the translational and rotational velocities of the human along the trajectory by using consecutive $\langle x, y \rangle$ samples. Next, we obtain a series of expected robot perceptions along this trajectory of logged $\langle x, y \rangle$ positions using the environment models.

The final controller is obtained by using the Narmax system identification technique to obtain an environment-model sensor-based controller. The computed human demonstrator's rotational velocity is used with the series of expected robot perceptions. The modelled laser perceptions are used as inputs to the Narmax

system with the computed rotational velocities of the human demonstrator as outputs. We used parameters with an input regression N_u of 10 and polynomial degree l of 2 to obtain the final Narmax model that had 96 terms.

Results In the final stage of the experiment the obtained final controller is used to drive the robot whilst the robot's actual positions during this stage of the experiment are logged as illustrated in Figure 10.

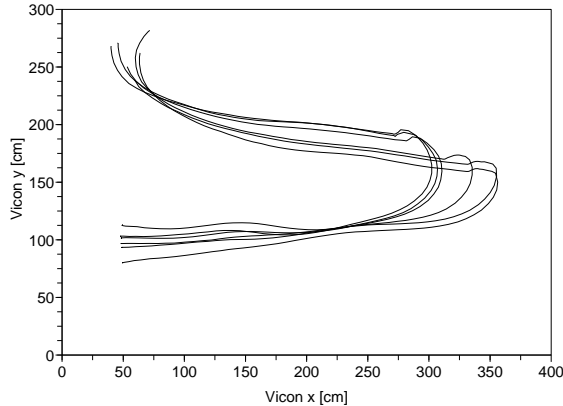


Figure 10: SIX TRAJECTORIES OF THE ROBOT UNDER CONTROL OF THE U-SHAPE SENSOR BASED CONTROLLER IN THE TEST ENVIRONMENT.

A visual inspection of Figure 10 shows that, although not perfect, the obtained controller produces a behaviour that resembles the human demonstrator's initial trajectory.

Summary and Conclusion

Summary

In this paper we present experiments that show that an agent's behaviour — in this case a human demonstrator — can be translated into directly-executable robot control code, meaning that no programming is at all necessary. The method of “translating” behaviour into code is shown in Figure 2.

Conclusions

There are a number of decisive advantages to the presented method of automatically generating robot-controlling code in this way:

1. Code generation is very quick,
2. The generated code is canonical, which allows the development of analysis tools such as sensitivity analysis methods,
3. The generated code is parsimonious, which is relevant when the code is to be used on robots with little on-board computing resources.

Future Work Our next experiments at the Intelligent Systems Research Centre here in Londonderry will address the following weaknesses of our current implementation:

1. The external-camera-based tracking system we used here is very precise, but also very expensive. In future we plan to use a camera-based tracking system, where the camera used is the one mounted on *Swilly*.
2. Developing models of higher accuracy, and
3. Developing smaller models.

Acknowledgements The authors gratefully acknowledge the support of the Leverhulme trust under grant number F00430F.

References

- Akanyeti, O., Nehmzow, U., Weinrich, C., Kyriacou, T., and Billings, S. (2007). Programming mobile robots by demonstration through system identification. In *European Conference on Mobile Robotics (ECMR)*, pages 162–167.
- Alissandrakis, A., Nehaniv, C., Dautenhahn, K., and Saunders, J. (2005). An approach for programming robots by demonstration: Generalization across different initial configurations of manipulated objects. In *Proceedings of the IEEE international symposium on computational intelligence in robotics and automation*, page 61.
- Billings, S. and Chen, S. (1998). The determination of multivariable nonlinear models for dynamical systems. In Leonides, C., (Ed.), *Neural Network Systems, Techniques and Applications*, pages 231–278. Academic press.
- Billings, S. and Voon, W. S. F. (1986). Correlation based model validity tests for non-linear models. *International Journal of Control*, 44:235–244.
- Coates, A., Abbeel, P., and Ng, A. (2008). Learning for control from multiple demonstrations. In *Proceedings of the 25th international conference on Machine learning*, pages 144–151. ACM.
- Demiris, Y. (2009). Knowing when to assist: developmental issues in lifelong assistive robotics. In *Proceedings of the 31st Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC 2009)*, pages 3357–3360, Minneapolis, Minnesota.
- Demiris, Y. and Dearden, A. (2005). From motor babbling to hierarchical learning by imitation: a robot developmental pathway. In Berthouze, L., Kaplan, F., Kozima, H., Yano, H., Konczak, J., Metta, G., Nadel, J., Sandini, G., Stojanov, G., and Balkenius, C., (Eds.), *Proceedings of the Fifth International Workshop on Epigenetic Robotics: Modeling Cognitive Development in Robotic Systems*, pages 87–93.
- Korenberg, M., Billings, S., Liu, Y. P., and McIlroy, P. J. (1988). Orthogonal parameter estimation algorithm for non-linear stochastic systems. *International Journal of Control*, 48:193–210.
- Nehmzow, U., Akanyeti, O., Weinrich, C., Kyriacou, T., and Billings, S. (2007). Robot programming by demonstration through system identification. In *IEEE/RSJ International Conference on Intelligent Robots and Systems, 2007. IROS 2007*, pages 801–806.

The Evaluation of AGI Systems

Pei Wang

Temple University, Philadelphia, USA
<http://www.cis.temple.edu/~pwang/>

Abstract

The paper surveys the evaluation approaches used in AGI research, and argues that the proper way of evaluation is to combine empirical comparison with human intelligence and theoretical analysis of the assumptions and implications of the AGI models.

Approaches of Evaluation

In recent years, the problem of evaluation is getting more and more attention in the field of Artificial General Intelligence, or AGI (GB09; LIIL09; LGW09; MAP⁺09; Was09). Though the evaluation of research results is important in any field of scientific research, the problem has special difficulty in the current context of AGI, since the research activities belong to many different paradigms, and there seems to be no “neutral” way to compare them (Wan08).

In traditional AI research, since the projects are typically problem-oriented, it is natural to compare competing theories and techniques by the scope, correctness, and complexity of the algorithms involved, or by their performance on a set of benchmark instances of the problem. Obviously, this methodology is no longer applicable to AGI, given its stress of generality. Very commonly, one technique performs better on one problem than another technique, but not as well on a second problem. In this case, how can we decide which technique is “generally better”?

One solution proposed in mainstream AI is to select typical intellectual problems as “Grand Challenges” (Coh05; Bra06). Though such activities do stimulate interesting research, it still has the danger of leading the research to problem-specific solutions, no matter how carefully the problems are selected — after all, this was why problems like theorem proving and game playing were selected in the early days of AI, and the resulting techniques have not been generalized to other fields very well.

An alternative is to use multiple tasks (GB09) as a kind of “Cognitive Decathlon” (MJMH07). This approach clearly covers a larger field than a single challenge problem, but the selection of the components for the compound problem, as well as the way to calculate the “total score”, may still look more or less arbitrary.

One way to be less problem-specific is to move away from testing problems, and to evaluate the systems according to certain properties (Min61; BBI⁺98; LIIL09). While providing important insights, this approach also needs to justify its selection of desiderata and its method of overall evaluation, especially since the proposed property lists are all different from each other, and few technique has all the properties.

Due to the lack of a common evaluation methodology, in papers surveying AGI what we usually find are descriptions of the special properties (both desired and undesired) of each system, without an overall grading or ranking (PG07; DOP08). Though this kind of description is often fair and informative, it does not resolve the evaluation problem, but avoids it to a large extent. For the field of AGI as a whole, there is a need for clear, justified, and widely applicable ways to evaluate research results, rather than treating them as equally good (though different).

Obviously, an evaluation should start from the research goal — without a clearly specified destination, it is simply impossible to compare who is closer to it. Unfortunately, here the lacking of a common goal is exactly the reason for the lacking of a common evaluation methodology. By comparing two AAAI Presidential Addresses a decade apart (Dav98; Bra06), it seems clear that mainstream AI as a whole is not moving closer to a consensus on what its research goal really is. This diversity in objective inevitably causes the research efforts to move in different directions (Wan08), and to apply different evaluation criteria.

At the most general level, AI has been driven by the motivations of understanding human intelligence, and reproducing it in computers to solve practical problems. Therefore, there are two fundamental approaches for evaluation:

The empirical approach: To evaluate the intelligence of a computer system according to its similarity to human intelligence.

The theoretical approach: To evaluate the intelligence of a computer system according to its agreement with a theory of intelligence.

Roughly speaking, they correspond to the “think/act

like human” and “think/act rationally” in (RN02), respectively; in terms of the five types of “AI” defined in (Wan08), the “Principle-AI” is associated more closely with theoretical evaluation, while the other four with empirical evaluation, in different ways.

In the rest of the paper, the two approaches are discussed, compared, and related to each other.

Empirical Evaluation

Since the best known form of intelligence is human intelligence, it seems obvious that the ultimate criterion used in the evaluation of AI should be how close a system is to human intelligence. Concretely, for a given AI system, “empirical evaluation” means to test the system in various situations, and then its “level of intelligence”, or some kind of “IQ”, is measured by how close its input/output data is to human data in similar situations, just like how a theory in natural science is usually evaluated.

However, in the context of AGI, the problem is much more complicated. Even if we all agree that AI can be evaluated using “human data”, there is still the question of *which* data should be used for this purpose. A brief survey of the related research shows that human intelligence has been studied at very different levels:

- There are researchers working on *brain* modeling, guided by the belief that the key of intelligence is hidden in the neural structure (dG07; HB04). However, even among them, there is still a huge difference on the type of data from neuroscience that is taken into consideration in the model.
- Starting from Turing (Tur50), there is the school that believes what really matters for intelligence are *behaviors*, and the intelligence of a computer system should be evaluated according to how the system acts like a human being. Similarly, in this school there are different opinions on what kind of behavior should be considered, and various variations of the Turing Test have been proposed (BBF01; Har00; MAP⁺09).
- There are psychologically motivated models of intelligence and cognition that focus on the *architecture* of the system, which is believed to be responsible for the production of the psychological data (AL98; Bac09; Fra07; New90). Among these architectures, again there are differences on which type of psychological data is considered.
- Many people judge the level of intelligence by the *problems* the system can solve, rather than by the details of the solving process. According to this kind of opinion, “general” (or “human-level”) intelligence means the capability of solving a wide range of problems as human beings (McC07). Therefore, to evaluate the intelligence of an AI system, we can test it with examinations used in primary and secondary schools (GB09), or to see how many human jobs it can be employed in to replace a human (Nil05).

There are very different answers to the “which data?” question, because human intelligence is a complicated phenomenon that has been studied at many different levels of description, and with focus on different aspects. All of these research paradigms are valid and valuable, but they lead the research to different directions (Wan08), and inevitably require different evaluation criteria.

The above situation may remind us of the well-known story of the blind men and an elephant (Was09). Why not to take all types of human data into consideration altogether when evaluating a model?

One reason is that each description of human intelligence is carried out at a specific “level”, using a vocabulary with specific scope and granularity. Consequently, it cannot be perfectly reduced or summarized into another level below or above. It is simply impossible to get a “complete description” of human intelligence that can satisfy all intellectual and practical purposes.

Even though in principle a computer can simulate any process in any accuracy, to duplicate human intelligence in this way is still an impossible practice, because to do that it is not enough to duplicate the neural electrical mechanism (since the other biological and chemical processes in the brain may be relevant), the complete brain (since the human body plays a significant role in cognition), or even a whole human being (since the influence of experience cannot be ignored in human behavior). Some people may argue that we can give the simulated human a simulated human experience in a simulated human world, but even in that scenario, the simulated world must be separated from our world, because we are not going to take a simulated human as a human, which means the simulated human cannot get our actual social experience.

Furthermore, even if such a complete simulation of human intelligence can be obtained, it is not what AI aims at. “Artificial Intelligence” is never an attempt to duplicate human intelligence *as it is*, in all aspects. Instead, it is the attempt to reproduce intelligence in computers, which are fundamentally different from the human brain at the hardware level. AI comes from the belief that “intelligence” is a phenomenon that may appear in various forms, and human intelligence is just one form of it. If computer intelligence can only be achieved via simulating human intelligence in all perceivable details, then the above belief will actually be falsified, rather than verified. Also, if “intelligence” merely means “human intelligence”, no room will be left for possibilities like “animal intelligence”, “group intelligence”, or “extra-terrestrial intelligence”. That would be a highly anthropocentric definition of intelligence. For the AI Dream to be fully realized, what to be created is a form of intelligence that is *similar* to human intelligence in certain essential aspects, but not in all aspects.

According to the above analysis, though it is possible, even necessary in certain sense, to evaluate AGI systems empirically by comparing them with human in-

telligence, the evaluation needs to be guided and justified by certain general principles, which decide the type of data to be considered in the evaluation. Though the human brain and mind is the major source of inspiration of AI research, it is not reasonable to judge the intelligence of an AGI system according to arbitrarily assembled human data.

Therefore, an empirical evaluation itself also needs justification and evaluation, and this “meta-evaluation” cannot be empirical, but must be theoretical. Before the behavior or performance of AGI systems are compared with certain human data, reasons must be given for why the type, scope, and granularity of the data are considered as directly related to aspects of human intelligence that can be meaningfully extended into machines, rather than as come from accidental factors that have little relevance to non-human intelligence. This is the case, because unlike models developed for pure biological, psychological, or evolutionary purposes, for AI/AGI models to be “just like human” may not be the ultimate aim.

Theoretical Evaluation

To many people, “intelligence” is the ability to find solutions that are correct and optimal in a certain sense; AI is the attempt to make computers do so, especially on problems that have not been solved in this sense; AGI means to achieve it in various domains by the same system.

With such an understanding of intelligence, the precondition for AI to be realized is a *theory* which specifies what is the correct or optimal solution to a problem, and how to find or build it. Such a theory must be general enough to be applied to various domains, and concrete enough to be implemented in a computer. Technically, it means the theory can be formalized. Clearly, it is a *normative theory* (like mathematics and computer science, which specify what should happen in a system to be created) of intelligence, rather than a *descriptive theory* (like biology and psychology, which specify what actually happens in a system to be understood).

The theoretical approach of evaluation means to compare the design and performance with the requirements and predictions of such a theory.

Though the above statement sounds reasonable, a question naturally follows: which theory? Currently there is no widely accepted “theory of intelligence”. Instead, researchers have been building systems on different theoretical foundations:

- One of the most influential traditions in AI is based on *mathematical logic* (Hay77; McC88; Nil91). In the current AGI research, there are also many projects where the basic ideas are to extend classical logic in various directions (Bri08; GKSK09; Pol08).
- Given the intrinsically uncertain nature of many problems in AI, probability theory has been proposed as a proper theoretical foundation (Che85; Pea88), which has been accepted by more and more

people in recent years. It leads to the belief that an AGI system, or at least a large part of it, should be designed using probability theory and statistical methods (GIGH08; Mil08).

- Since AI systems are eventually implemented in computers, there is no surprise that many works are based on theory of computation, by analyzing problem-solving processes in terms of algorithm, computability, and computational complexity (Mar77; HF95). The same methodology is often used in the design and discussion of AGI systems (Bau04; Sch07).
- There have been various attempts to develop new normative theories specially for intelligence (Alb91; Hut05; Kug04; Rus97; Wan06). Each of the theories has some similarity with the traditional theories, but also introduces new postulates and assumptions, according to the beliefs of the researcher.

Since different theories usually give different evaluation results, once again we are facing a “meta-evaluation” problem: before AGI projects can be evaluated according to a theory, the theory itself needs to be analyzed and evaluated.

For our current purpose, a normative theory should be evaluated in two aspects: its intrinsic merit and its applicability to the AGI problem. Such a theory is based on a set of postulates and assumptions, from which the theorems and conclusions are derived by justified rules and procedures. Since the traditional theories (mathematical logic, probability theory, and theory of computation) have been thoroughly studied, the major problem about them is not in themselves, but in whether they can solve the problem of AGI.

Probably few people will insist that one of the traditional theories, in its canonical form, is sufficient for AGI. Instead, every “traditional-theory-based” AGI project has more or less extended and/or modified the theory, so as to make its assumptions and requirements satisfiable, and their conclusions competent to solve the problems in AGI. For instance, classical logic has been modified in various ways to deal with many types of uncertainty; probability theory is usually supplemented with assumptions on independence of the variables and the type of the distribution; theory of computation is often applied with simplifications and approximations here or there.

In this process, an important issue is the validness of the modification — to what extent a theory can be modified while remaining itself? Usually, new assumptions can be added, as far as they do not conflict with the existing ones. However, this is often not enough, and some postulates and axioms of the theory may be dropped or modified. One representative case is the reasoning model Probabilistic Logic Network, or PLN (GIGH08). When introducing it, the authors declare that “while probability theory is the foundation of PLN, not all aspects of PLN are based strictly on probability theory” (GIGH08, page 4). For instance, since “it is too much to expect any severely resource-constrained intel-

ligence to be fully self-consistent” (GIGH08, page 53), the consistency axiom of probability theory is dropped, and the system may assign different probability values to the same conclusion when following different reasoning paths. Though the reason is quite understandable, its effect needs clarification — should PLN be referred to as a revision of a probabilistic model of reasoning, or a model that is merely similar to a probabilistic model? When an axiom of probability theory has been violated, are the theorems of the theory (such as Bayes’) still valid? Why? Answers to these questions can help us to decide whether PLN can be validated by its relation with probability theory.

The applicability problem is also a subtle one. Every normative theory is an idealization, and its applications into concrete domains are almost always rough, rather than accurate. If one assumption of the theory cannot be fully satisfied by the actual situation in a domain, will the theory become completely irrelevant, or remain an idealized case to be approximated? Such a situation can be found in (LH07), which defines the concept of “universal intelligence” as the ability to provide optimal solutions to problems, formalized in a theoretical model described in (Hut05). One notable assumption of the model is that it requires unlimited time-space resources, and the authors “consider the addition of resource limitations to the definition of intelligence to be either superfluous, or wrong” (LH07). Though it is well known that people in AI have very different understandings about “intelligence”, it is obvious that “optimization with resource restriction” and “optimization without resource restriction” lead to very different models, though they are all “optimal”, in certain (different) sense. If no concrete intelligent system can have unlimited resources, to what extent can these systems be properly evaluated according to a standard based on the opposite assumption?

To take the theoretical approach in evaluating AGI systems, we first need a well-established normative theory, with clearly stated assumptions, and conclusions implied by the assumptions. Furthermore, the assumptions should be satisfied by human intelligence, which is not only the best example of intelligence, but also widely believed to be selected by the evolution process, so is optimal in certain sense. For the same reason, the theory should be adequate in explaining various features of human intelligence that are desired to be reproduced in computers. Such a task should be carried out by comparing the theory with the reality of human intelligence. When approximation and simplification are needed, they should not completely change the nature of the problem. Otherwise, the theory cannot be convincingly used in the evaluation of systems — no matter how excellent the theory is on its own, it may not be applicable to the problem of AGI.

Therefore, a theoretical evaluation itself also needs justification and evaluation, and this “meta-evaluation” cannot be theoretical, but must be empirical. Before the design or performance of AGI systems are compared

with what is required or predicted by a theory, reasons must be given to argue that the theory is at least satisfied by human intelligence. Otherwise the theory should not be designated as a “theory of intelligence”.

Conclusion and Application

To establish widely applicable evaluation criteria is an important task for the field of AGI. It will not only enable systems and projects to be compared, but also guide the research in correct directions. Though in the near future the field will continue to host multiple research paradigms (Wan08), it is nevertheless necessary to avoid misleading evaluation approaches.

Given the nature of the AGI problem, there is no “natural” or “self-evident” way to evaluate the research. Any evaluation standard needs to be justified to be proper for the task. An evaluation proposal may be well-motivated, but still leads the research to a undesired direction by making an improper demand.

Compared to selected practical problems or functional features, it is more justifiable to evaluate an AGI system empirically according to human data, or theoretically according to an optimization model. In either case, the “meta-evaluation” is more general, reliable, consistent, and nonarbitrary.

The empirical approach of evaluation takes an AGI system as a descriptive model of human intelligence, made at an abstract level so that it becomes implementable in computers. For this approach, the meta-evaluation should take the form of a theoretical analysis, to argue that the selected data not only capture regularities in *human* intelligence, but also in other forms of intelligence. The major challenge in this approach is to separate the “intelligence-general” aspects of human intelligence from the “human-specific” aspects. The most likely mistake here is to propose a highly anthropocentric standard for AGI, which, even if possible to be achieved, will limit our imagination and innovation, and restrict the research in unnecessary ways resulting in “Artificial Human Intelligence”, rather than “Artificial (General) Intelligence”.

The theoretical approach of evaluation takes an AGI system as a normative model of intelligence that captures the essence of human intelligence at an abstract level. For this approach, the meta-evaluation should take the form of an empirical justification of the assumptions of the model (that is, they are indeed satisfied by human intelligence) and the model’s explanatory power (that is, it is accountable for the cognitive functions observed in human intelligence). The major challenge in this approach is to identify the basics of human intelligence and to express them in a computer-implementable way. The most likely mistake here is to propose a highly biased standard for AGI, which, even if possible to be achieved, will lack key characteristics of intelligence as we commonly know, and to lead the research on deviant paths resulting in “Artificially Designated Intelligence”, rather than “Naturally Designated Intelligence”.

Now we see that the empirical approach and theoretical approach of evaluation actually depend on each other for the meta-evaluation. No matter which approach is selected, the other one will also be needed, though the two will serve different purposes in the whole evaluation process.

To make the above conclusion more concrete, let us briefly see how it is applied to the evaluation of the author's own research project, NARS (Wan06).

NARS is based on the belief that "intelligence" is the capability of *adaptation with insufficient knowledge and resources*. This belief itself is justified empirically — the human mind does have such capability (MR92).

The theory of intelligence built on this belief is a normative one, that is, it specifies how an intelligent system should work, not restricted by the biological, psychological, or evolutionary details of human intelligence. It is a theory of optimization, in the sense that if a system has to live and work in an environment, where the future cannot be accurately predicted, and the system's time-space resources are usually in short supply, then the theory provides the best design for the system. All the major design decisions of NARS are justified by theoretical analysis with respect to this objective, rather than by duplicating the human counterparts as faithfully as possible.

This theory is different from the traditional theories (classical logic, probability theory, theory of computation, etc.), mainly because of the above basic assumption. Since none of the traditional theories was developed for the problem of general intelligence, they do not assume the necessity of adaptation, nor the insufficiency of knowledge and resources in all aspects. Because this assumption plays a fundamental role in the theory, the issue cannot be resolved by minor extensions and modifications. Consequently, the NARS theory of intelligence is not based on any previous theory, though it surely inherits many ideas from them, and still uses them for subproblems here or there.

Even though NARS is primarily evaluated by theoretical analysis, to compare the performance and properties of the system with "human data" still makes sense. Since the human mind is the solution found by evolution for the same problem, it is not a coincidence that a truly intelligent AI system should share many similar properties with the human mind. For example, it should depend on certain learning mechanisms to deal with the uncertain future, while managing its own resources to achieve the best overall efficiency. However, due to its fundamental non-human hardware and experience, there is no reason to expect NARS to reproduce the human data exactly. "How the human mind does it" is a source of inspiration for the design decisions, but not their direct justification. For a new theory under development, empirical testing also reveals implicit and hidden implications of the assumptions — if NARS does something fundamentally different from human beings, then an explanation will be required, though it does not necessarily lead to a revision of the model.

In summary, to evaluate AGI systems, we need to properly combine the empirical approach and the theoretical approach, so as to find an identity for AGI that is neither too close to human intelligence (to become *Artificial Human Intelligence*), nor too far away from it (to become *Artificially Designated Intelligence*).

Acknowledgments

Thanks to Jeff Thompson for helpful comments and English corrections.

References

- John R. Anderson and Christian Lebiere. *The Atomic Components of Thought*. Lawrence Erlbaum Associates, Mahwah, New Jersey, 1998.
- James S. Albus. Outline for a theory of intelligence. *IEEE Transactions on Systems, Man, and Cybernetics*, 21(3):473–509, 1991.
- Joscha Bach. *Principles of Synthetic Intelligence PSI: An Architecture of Motivated Cognition*. Oxford University Press, Oxford, 2009.
- Eric B. Baum. *What is Thought?* MIT Press, Cambridge, Massachusetts, 2004.
- Selmer Bringsjord, Paul Bello, and David Ferrucci. Creativity, the Turing Test, and the (better) Lovelace Test. *Minds and Machines*, 11(1):3–27, 2001.
- Rodney A. Brooks, Cynthia Breazeal, Robert Irie, Charles C. Kemp, Matthew Marjanovic, Brian Scassellati, and Matthew M. Williamson. Alternative essences of intelligence. In *Proceedings of the Fifteenth AAAI/IAAI Conference*, pages 961–968, 1998.
- Ronald J. Brachman. (AA)AI — more than the sum of its parts, 2005 AAAI Presidential Address. *AI Magazine*, 27(4):19–34, 2006.
- Selmer Bringsjord. The logicist manifesto: At long last let logic-based artificial intelligence become a field unto itself. *Journal of Applied Logic*, 6(4):502–525, 2008.
- Peter Cheeseman. In defense of probability. In *Proceedings of the Eighth International Joint Conference on Artificial Intelligence*, pages 1002–1009, 1985.
- Paul R. Cohen. If not Turings Test, then what? *AI Magazine*, 26:61–67, 2005.
- Randall Davis. What are intelligence? and why? 1996 AAAI Presidential Address. *AI Magazine*, 19(1):91–111, 1998.
- Hugo de Garis. Artificial brains. In Ben Goertzel and Cassio Pennachin, editors, *Artificial General Intelligence*, pages 159–174. Springer, Berlin, 2007.
- Włodzisław Duch, Richard Oentaryo, and Michel Pasquier. Cognitive architectures: where do we go from here? In *Proceedings of the First Conference on Artificial General Intelligence*, pages 122–136, 2008.
- Stan Franklin. A foundational architecture for artificial general intelligence. In Ben Goertzel and Pei

- Wang, editors, *Advance of Artificial General Intelligence*, pages 36–54. IOS Press, Amsterdam, 2007.
- Ben Goertzel and Stephan Vladimir Bugaj. AGI Preschool: a framework for evaluating early-stage human-like AGIs. In *Proceedings of the Second Conference on Artificial General Intelligence*, pages 31–36, 2009.
- Ben Goertzel, Matthew Iklé, Izabela Freire Goertzel, and Ari Heljakka. *Probabilistic Logic Networks: A Comprehensive Framework for Uncertain Inference*. Springer, New York, 2008.
- Helmar Gust, Ulf Krumnack, Angela Schwering, and Kai-Uwe Kühnberger. The role of logic in AGI systems: towards a lingua franca for general intelligence. In *Proceedings of the Second Conference on Artificial General Intelligence*, pages 43–48, 2009.
- Stevan Harnad. Minds, machines and Turing: the indistinguishability of indistinguishables. *Journal of Logic, Language, and Information*, 9:425–445, 2000.
- Patrick J. Hayes. In defense of logic. In *Proceedings of the Fifth International Joint Conference on Artificial Intelligence*, pages 559–565, 1977.
- Jeff Hawkins and Sandra Blakeslee. *On Intelligence*. Times Books, New York, 2004.
- Patrick Hayes and Kenneth Ford. Turing Test considered harmful. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, pages 972–977, 1995.
- Marcus Hutter. *Universal Artificial Intelligence: Sequential Decisions based on Algorithmic Probability*. Springer, Berlin, 2005.
- Peter Kugel. Toward a theory of intelligence. *Theoretical Computer Science*, 317(1-3):13–30, 2004.
- Christian Lebiere, Cleotilde Gonzalez, and Walter Warwick. A comparative approach to understanding general intelligence: predicting cognitive performance in an open-ended dynamic task. In *Proceedings of the Second Conference on Artificial General Intelligence*, pages 103–107, 2009.
- Shane Legg and Marcus Hutter. Universal intelligence: a definition of machine intelligence. *Minds & Machines*, 17(4):391–444, 2007.
- John E. Laird, Robert E. Wray III, Robert P. Marinier III, and Pat Langley. Claims and challenges in evaluating human-level intelligent systems. In *Proceedings of the Second Conference on Artificial General Intelligence*, pages 91–96, 2009.
- W. Joseph MacInnes, Blair C. Armstrong, Dwayne Pare, George S. Cree, and Steve Joordens. Everyones a critic: memory models and uses for an artificial Turing judge. In *Proceedings of the Second Conference on Artificial General Intelligence*, pages 132–137, 2009.
- David Marr. Artificial intelligence: a personal view. *Artificial Intelligence*, 9:37–48, 1977.
- John McCarthy. Mathematical logic in artificial intelligence. *Dædalus*, 117(1):297–311, 1988.
- John McCarthy. From here to human-level AI. *Artificial Intelligence*, 171:1174–1182, 2007.
- Brian Milch. Artificial general intelligence through large-scale, multimodal Bayesian learning. In *Proceedings of the First Conference on Artificial General Intelligence*, pages 248–255, 2008.
- Marvin Minsky. Steps towards artificial intelligence. *Proceedings of the Institute of Radio Engineers*, 49:8–30, 1961.
- Shane T. Mueller, Matt Jones, Brandon S. Minnery, and Julia M.H. Hiland. The BICA Cognitive Decathlon: A test suite for biologically-inspired cognitive agents. In *Proceedings of the Behavior Representation in Modeling and Simulation Conference*, 2007.
- Douglas L. Medin and Brian H. Ross. *Cognitive Psychology*. Harcourt Brace Jovanovich, Fort Worth, 1992.
- Allen Newell. *Unified Theories of Cognition*. Harvard University Press, Cambridge, Massachusetts, 1990.
- Nils J. Nilsson. Logic and artificial intelligence. *Artificial Intelligence*, 47:31–56, 1991.
- Nils J. Nilsson. Human-level artificial intelligence? Be serious! *AI Magazine*, 26(4):68–75, 2005.
- Judea Pearl. *Probabilistic Reasoning in Intelligent Systems*. Morgan Kaufmann Publishers, San Mateo, California, 1988.
- Cassio Pennachin and Ben Goertzel. Contemporary approaches to artificial general intelligence. In Ben Goertzel and Cassio Pennachin, editors, *Artificial General Intelligence*, pages 1–30. Springer, New York, 2007.
- John Pollock. OSCAR: an architecture for generally intelligent agents. In *Proceedings of the First Conference on Artificial General Intelligence*, pages 275–286, 2008.
- Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, Upper Saddle River, New Jersey, 2nd edition, 2002.
- Stuart Russell. Rationality and intelligence. *Artificial Intelligence*, 94:57–77, 1997.
- Jürgen Schmidhuber. The new AI: General & sound & relevant for physics. In Ben Goertzel and Cassio Pennachin, editors, *Artificial General Intelligence*, pages 175–198. Springer, Berlin, 2007.
- Alan M. Turing. Computing machinery and intelligence. *Mind*, LIX:433–460, 1950.
- Pei Wang. *Rigid Flexibility: The Logic of Intelligence*. Springer, Dordrecht, 2006.
- Pei Wang. What do you mean by “AI”? In *Proceedings of the First Conference on Artificial General Intelligence*, pages 362–373, 2008.
- Mark Waser. What is artificial general intelligence? Clarifying the goal for engineering and evaluation. In *Proceedings of the Second Conference on Artificial General Intelligence*, pages 186–191, 2009.

Concept Formation in the Ouroboros Model

Knud Thomsen

Paul Scherrer Institute, CH-5232 Villigen PSI, Switzerland

Abstract

According to the Ouroboros Model several occasions can be distinguished over the course of the general autonomous cyclic activity in which new concepts are established and associated memories are preferentially laid down. Whereas a rather standard habituation process can lead to the extraction of statistical regularities from repeated common (consecutive) excitation, two specific instances are peculiar to the Ouroboros Model; the consumption analysis process marks events when especially successful or the contrary. In addition, new concepts can be assembled very quickly by combining previously existing building blocks. Relations of these theoretical considerations to supporting recent experimental findings are briefly outlined.

The Ouroboros Model in a Nutshell

The Ouroboros Model proposes a novel algorithmic architecture for efficient data processing in living brains and for artificial agents [1]. At its core lies a general repetitive loop where one iteration cycle sets the stage for the next. All concepts of an agent are claimed to be organized into hierarchical data structures called schemata, also known as frames, scripts or the like.

During perception, any sensory input activates schemata with equal or similar constituents encountered before, thus expectations for the whole unit and especially for its parts are kindled. This corresponds to the highlighting of empty slots in the selected schema as biasing anticipated features facilitates their actual excitation. These predictions are subsequently compared to the factually encountered input. Depending on the outcome of this "consumption analysis" different next steps are taken. In case of partial fit search for further data continues; a reset, i.e. a new attempt employing another schema, is triggered if the occurring discrepancies are too big. Basically the same process applies for all other actions like memory search, active movements of the agent or language production.

Self-referential monitoring of the whole process, and in particular of the flow of activation directed by the consumption analysis, yields valuable feedback for the optimum allocation of attention and resources including the selective establishment of useful new concepts.

According to the Ouroboros Model basically four different situations in which novel concepts are formed and corresponding fresh memory entries are first created and shaped can be distinguished.

Ways to Concept Formation

Two types of occasions are directly marked in the Ouroboros Model as interesting by the outcome of the consumption analysis, and preferentially for them new records are laid down:

- Events, when everything fits perfectly; i.e. associated neural representations are stored as kind of snapshots of all concurrent activity, making them available for guidance in the future as they have proved useful once.
- Constellations, which led to an impasse, are worthwhile remembering, too; in this case for future avoidance.

These new memories stand for junks, i.e. concepts, again as schemata, frames or scripts. Their building blocks include whatever representations are active at the time when the "snapshot" is taken, including sensory signals, abstractions, previously laid down concepts, and emotions. They might but need not include / correspond to a direct representation unit like a word. At later occasions they will serve for controlling behavior, by guiding action to or away from the marked tracks.

Knowledge thus forms the very basis for the data processing steps, and its meaningful expansion is a prime outcome of its use as well; the available data base of concepts / schemata is steadily enlarged and completed, especially in areas where the need for this surfaced and is felt most strongly.

Even without the strong motivation by an acute alert signal from consumption analysis novel categories and concepts are assembled on the spot:

- New concepts are built from existing structures

We can quickly establish new compound concepts, whole scenes, from previously existing building blocks, i.e. by combining (parts of) other concepts; here is an example: Let us assume that we hear about "the lady in the fur coat". Even without any further specification a figure is defined to a certain extent including many implicit details. Also in case we heard this expression for the first time the concept

is established well enough for immediate use in a subsequent cycle of consumption analysis, expectations are effectively triggered. When we now see a woman in this context, we are surprised if she is naked on her feet (...unless she is walking on a beach). Fur coats imply warm shoes or boots, unless the wider frame already deviates from defaults.

In parallel to the above described instant establishing of concepts and the recording of at least short time episodic memory entries there exists a slower and rather independent process:

- Associations and categorizations are gradually distilled from the statistics of co-occurrences.

In the sense, that completely disadvantageous or fatal activity would not be repeated many times, also this grinding-in of associations can be understood as a result of successful or even rewarded activations.

Activity, which forms the basis of this comparatively slow process can pertain to many different representations starting from low level sensory signals to the most abstracts data structures already available, and of course, their combination.

Relation to Recent Experimental Findings

The most important ingredient in the Ouroboros Model is the repetitive and self-reflective consumption analysis process. A key conjecture derived from this algorithmic structure is the highlighting of interesting occasions and the quick recording of corresponding memories for advantageous future use. The Ouroboros Model proposes to distinguish “index-entries” as pointers to the “main text” containing more details. On the basis of a wealth of experimental data, a similar general division of work in the mammalian brain has been proposed some time ago [2]. Hippocampal structures are well suited for fast recording of distinct entries, they are thought to link memories spread widely over the cortex, where minute details are memorized on longer time scales.

Dopamine signals are widely considered to act as highlighting behaviorally important events, midbrain dopamine neurons code discrepancies between expected and actual rewards [3].

If dopamine now is the best established marker for discrepancies and if associated constellations should lead to the immediate recording of new concepts, at least of their specific index entry, one would expect that dopamine release has a profound impact on hippocampal long term potentiation, generally accepted as a decisive substrate for memories. This now is exactly what has been found just recently: dopaminergic modulation significantly increases sensitivity at hippocampal synapses [4]. In addition, temporal contrast is lost, i.e. not only consecutive activations lead to enhancement, but also activations in

reverse order, which normally result in an attenuation of a connection. Thus, a memory entry is established, which connects in an encompassing snapshot all activity associated with the occurrence of a dopamine burst.

Along with the enhanced storage of “index entries”, the preferential establishment of traces in the “text” occurs. In the cortex, several neuromodulator systems, in particular widespread cholinergic innervation, have been conjectured to control attention and associative learning under the control of error driven learning mechanisms [5].

The Ouroboros Model holds that in the brain often several mechanisms working to the same end are implemented in parallel.

Given the demanding boundary conditions, in particular, the stringent time constraints, for any actor in the real world, not all memories are of equal value or even sorted out to the same degree. Incompletely processed information has been claimed to be discarded off-line in living brains while sleeping and dreaming [6].

During the process of clearing the brain of not (yet) useful remainders, sleeping and dreaming might still serve to prime associative networks [7]. Unassociated but otherwise well-established information has been found to be integrated into associative networks, i.e. schema structures, after REM (rapid eye movement) sleep.

Obviously, much work is still required to establish detailed relations as suggested by the Ouroboros Model.

References

- [1] K. Thomsen, “The Ouroboros Model”, BICA 08, Technical Report FS-08-04, Menlo Park, California: AAAI Press, 2008.
- [2] R. C. O’Reilly and J. W. Rudy, “Computational Principles of Learning in the Neocortex and Hippocampus”, *Hippocampus* 10, 389-397, 2000.
- [3] W. Schultz, “The Reward Signal of Midbrain Dopamine Neurons”, *News Physiol. Sci.* 14, 249-255, 1999.
- [4] J-C. Zhang, P.M Lau, and G.Q Bi, “Gain in sensitivity and loss in temporal contrast of STDP by dopaminergic modulation at hippocampal synapses”, *PNAS* 106, 13028-13033, 2009.
- [5] W. M. Pauli and R. C. O’Reilly, “Attentional control of associative learning – A possible role of the cholinergic system”, *Brain Research* 1202, 43-53, 2008.
- [6] K. Thomsen, “The Ouroboros Model”, *Cogprints* 6081, <http://cogprints.org/6081/>, 2008.
- [7] D. J. Cai, S. A. Mednick, E. M. Harrison, J. C. Kanady, and S. C. Mednick, “REM, not incubation, improves creativity by priming associative networks”, *PNAS* 106, 10130-10134, 2009.