

**2013 Annual Conference on Advances in Cognitive Systems:
Workshop on Metacognition in Situated Agents**

Computer Science Technical Report No. CS-TR-5030

UMIACS Technical Report No. UMIACS-TR-2013-07

Darsana Josyula (co-chair), Bowie State University

Paul Robertson (co-chair), Doll Labs

Michael T. Cox, University of Maryland Institute for Advanced Computer Studies

College Park, MD 20742

Preface

Metacognition is the process of thinking about thinking. It provides cognitive systems the ability to note and deal with anomalies, changes, opportunities, surprises and uncertainty. It includes both monitoring of cognitive activities and control of such activities. Monitoring helps to evaluate and explain the cognitive activities, while control helps to adapt or modify the cognitive activities.

Situated agents are agents embedded in a dynamic environment that they can sense or perceive and manipulate or change through their actions. Similarly, they can act in order to manipulate other agents among which they are situated. Examples might include robots, natural language dialog interfaces, web-based agents or virtual-reality bots.

An agent can leverage metacognition of its own thinking about other agents in its situated environment. It can equally benefit from metacognition of the thinking of other agents towards itself. Metacognitive monitoring can help situated agents in negotiations, conflict resolution and norm-awareness. Metacognitive control can help coordination and coalition formations of situated social agents.

In this workshop, we investigate the monitoring and control aspects of metacognition about self and other agents, and their application to situated artificial agents. The papers in this report cover some of the current work related to metacognition in the areas of meta-knowledge representation, meta-reasoning and meta-cognitive architecture.

Perlis et al. outlines a high-level view of architectures for real-time situated agents and the reliance of such agents on metacognition. Mbale, K. and Josyula, D presents a generic metacognitive component based on preserving the homeostasis of a host agent. Pickett, M. presents a framework for representing, learning, and processing meta-knowledge. Riddle, P. et al. discusses meta-level search through a problem representation space for problem–reformulation. Caro, M et al. uses metamemory to adapt to changes in memory retrieval constraints. Langley, P. et al. abstracts general problem specific abilities into strategic problem solving knowledge in an architecture for flexible problem solving across various domains. Samsonovich, A. examines metacognition as a means to improve fluid intelligence in a cognitive architecture. Perlis, D. and Cox, M. discuss the application of metacognitive monitoring to anomaly detection and goal generation.

Darsana Josyula and Paul Robertson
Baltimore, MD
14 December 2013

Table of Contents

Preface..... iii

Don Perlis, Michael T. Cox, Michael Maynard, Elizabeth McNany, Matthew Paisner, Vikas Shivashankar, Emily Hand, Jared Shamwell, Tim Oates, Tongchun Du, Darsana Josyula and Manuel Caro

A broad vision for intelligent behavior: perpetual real-world cognitive agents 1

Manuel Caro, Darsana Josyula and Jovani Jimenez

Metamemory for Information Retrieval from Long-term Memory in Artificial Cognitive Systems 17

Kenneth Mbale and Darsana Josyula

General Purpose Metacognition Engine 35

Don Perlis and Michael T. Cox

Autonomy beyond Anomalies and Goals: A Strategic Perspective..... 51

Marc Pickett

Towards A Unified Framework for Learning and Processing Perceptual, Relational, and Meta Knowledge..... 55

Pat Langley, Miranda Emery, Michael Barley and Christopher Maclellan

An Architecture for Flexible Problem Solving..... 65

Patricia Riddle, Mike Barley and Santiago Franco

Preliminary Results on a Meta-Level Search Framework for Problem Reformulation8Error! Bookmark not defined.

Alexei Samsonovich

How to Develop Fluid Intelligence via Metacognitive Self-Organization..... 99

A Broad Vision for Intelligent Behavior: Perpetual Real-World Cognitive Agents

Don Perlis PERLIS@CS.UMD.EDU
Michael T. Cox MCOX@CS.UMD.EDU
Institute for Advanced Computer Studies, University of Maryland, College Park, MD 20742 USA

Michael Maynard MAYNORD@UMD.EDU
Elizabeth McNany BETH@CS.UMD.EDU
Matthew Paisner PAISNER@CS.UMD.EDU
Vikas Shivashankar SVIKAS@CS.UMD.EDU
Emily Hand EMHAND@CS.UMD.EDU
Computer Science Department, University of Maryland, College Park, MD 20742 USA

Jared Shamwell EJSHAM@UMD.EDU
Neuroscience and Cognitive Science Program, University of Maryland, College Park, MD 20742 USA

Tim Oates OATES@CS.UMBC.EDU
Department of Computer Science and Electrical Engineering, University of Maryland, Baltimore County, Baltimore, MD 21250 USA

Tongchun Du TONGCHUNDU@GMAIL.COM
Department of Automation, Harbin Engineering University, Harbin, China

Darsana Josyula DARSANA@CS.UMD.EDU
Computer Science Department, Bowie State University, Bowie, MD 20715 USA

Manuel Caro MFCAROP@UNAL.EDU.CO
Department of Educational Informatics, University of Cordoba, Monteria, Colombia

Abstract

We describe ongoing work toward automating human-level behavior that pulls together much of traditional artificial intelligence in a real-time robotic setting. Natural-language dialog, planning, perception, locomotion, commonsense reasoning, memory, and learning all have key roles in this; and metareasoning is a sort of glue to guide the robot through rough spots.

1. Introduction

We are in the middle of a multi-year research program aimed at pulling together many parts of artificial intelligence in a suitable manner so that an agent constructed along such lines may come closer to “human-level” performance. The target performance spans not only a wide range of specific tasks, but also includes the ability to learn about, adapt to, and adjust both its environment and its own abilities (not unlike a human baby). While strict mimicking of human behavior is not our goal, we are mindful that we have much to learn from human behavior, and do not hesitate to make use of both established and intuitive insights therein. See Shamwell et al. (2012) for more along these lines.

Thus learning/adaptation occupies a special place in our vision. But so do general reasoning, perception, and action. Indeed, one theme that is central to our effort is that a system can and should be able to learn about the effects of its own actions via a combination of perception, inference and planning. Not only that, but a particular form of inference – metareasoning – is critical for us. For instance, such an agent may decide (by reasoning about its past episodes of reasoning) that it is not very good at solving certain kinds of problems and may then ask for help in learning to deal with such a problem. That in turn may involve natural language processing, and so on.

This paper is organized as follows: We will begin with a running example to illustrate much of our ideas; then we will focus separately on some of the pieces that the example depends on, where we already have had some successes; we then give a brief review of related work; and finally we will describe our current and future plans. The majority of this paper is devoted to the elements that we believe are essential to autonomous agents that must operate in the real world. Both reasoning and meta-reasoning fall into this category, with our approach to the latter involving a time-situated logic called active logic and a lightweight, general purpose architecture for meta-reasoning called the Meta-Cognitive Loop. Because we want our agents to engage with humans, we describe our experience with a dialog system based on active logic called ALFRED. A discussion of reinforcement learning follows that, given the prominent role that it plays in (low-level) learning to interact with the world. Next we explore an often overlooked aspect of autonomy - goal generation - and discuss it in the context of an overall cognitive architecture called MIDCA, and finish with a discussion of how memory interacts with all of these components.

We offer a cautionary note: while our main example involves a robot, we are not here suggesting a project in traditional robotics at all. Rather our aim is along the lines of the original conception of AI as the computational study of intelligent human-level behavior (see Langley, 2012). Thus swarm robotics, multi-agent systems, and the like, while important in their own right, have little to do with what we are investigating here. The science-fiction image of a household robot that can do many things while learning on the job is closer to our vision; and to those who say this is not where big successes are to be found, we reply: that is the issue, and we are offering to test it in what we think is a new way. Indeed, if this cannot be done, then it would seem that human-level intelligent behavior is not largely computational after all, and that in itself would be big news.

2. Running Example

Consider a robot – Robbie – whom we have assigned the task of obtaining a particular book and bringing it to us. We have told Robbie to look for the book in room 128, and that we need the book before noon. It is now 11:30am. Robbie sets off for room 128, having previously learned a floor

plan that she now consults to plan a path. She also marks the task details so that they remain in working memory; she knows from past experience that without this precaution she can lose sight of details and it can take lots of time to recover them, and that in this case time matters.

We pause here to give some background about Robbie. She is one year old, having had that much near-continuous training since being first turned on (and having stored her acquired knowledge appropriately in various forms of memory). That is, Robbie is a perpetual agent: she has a lifetime of her own and is not simply turned on briefly when we want her to do something. At first, she knew very little – just what the factory had installed in her KB – and even less about how to perform physical actions with any dexterity. But she has an internal goal of sorts, based on the maxim that “knowledge is good.” So when she is not working on an assigned (exogenous) task she pursues the endogenous goal of trying to learn about whatever she can. Of course, that can quickly lead to disaster since some explorations can lead to injury to herself and others; so she also has some initial cost-benefit information which she augments and modifies as she learns from experience including human interaction. In addition, she has had to learn to “see” – that is, to interpret her perceptual data-flow – and to relate that to her own activity: if she moves forward, her visual flow changes in one way; if she rotates, it changes in another; and if she reaches forward she sees her own arm, etc. She also has had to learn what books are: how they look, that they can be picked up, and so on. All this has taken up much of her first year.

To render the present context a bit more fully: we are studying for an exam that will take place at 12:30pm that same day; the book might possibly be helpful but we are fully occupied in memorizing some key items and cannot take the time to get the book ourselves; and we must set out for the exam by noon. Now we return to our story: Robbie is looking for a book...

Along the way, Robbie encounters a cluttered 10m section of hall, which slows her down considerably. After some time spent in trying to pass through that section, she decides instead to go on another, longer, route that, she anticipates, is not cluttered.

On arriving at the intended room, Robbie expects to find that it has the number 128 but instead observes 123. She puzzles about this and checks the doors on both sides, which are numbered 127 and 129. She looks again at the 123 and supposes that it either is a mistake or that the 8 has somehow degraded and now looks like a 3, and that in any case she is indeed at the correct room. She looks at the suspect 3 more carefully, detects what might have been the degraded part of an 8, and makes note of this for future use in reading numbers.

But the door to the room is closed. Robbie tries the knob, and cannot manage to turn it, having been trained only on door handles, not knobs. She makes numerous attempts in different ways, using one or more of her problem-solving algorithms, but still fails.

Robbie decides she needs help, and comes back to us for advice. It is now 11:45. We tell her that we did not realize that the door to room 128 was closed, and that knobs are too hard for her grippers to manage, but if she uses a key, the door will open without having to turn the knob at all. We give her a key, she starts off, and arrives again at room 128. After some effort the key allows entry, and as a result, Robbie learns a new method for opening the door.

However once inside she discovers that the book is not at the shelf position where it should be. It must have been misfiled, and there are hundreds of books to look through (the room seems to be a library of sorts). So she realizes that with all the time already taken – it is now 11:55 – it is very unlikely she will be able to find the book and still get it to us by noon. She returns and tells us this. We agree and tell her not to bother after all, and thank her for her efforts.

A great many things happen in this example. Yet they can be broken down into pieces most of which, separately, have been the subjects of enormous amounts of (highly successful) research. On the other hand, putting it all together effectively is far more than merely assembling the pieces. New issues arise – or become far more important – in the aggregate; for example, the need to know (and reason about) one’s own knowledge and behavior; the need to keep track of ongoing time; the need to mix plans and speech and others’ interests all in the same bit of reasoning; the need to recognize when events are anomalous; the need to notice salient features of events for future use; and the need to manage memory efficiently so as to keep relevant information in working memory and irrelevant information out.

This example may seem absurdly difficult, far beyond what anyone has any hope of achieving in the next several decades. But we think not. In the following several sections, we will describe work we have done on a number of topics closely related to the above kind of scenario. We start with the need to reason about one’s own knowledge in the face of contradictory information, and to monitor time-passage during reasoning.

3. Reasoning and Metareasoning: Active Logic

Active logics (Anderson, Josyula, Okamoto, & Perlis, 2002) are a family of formalisms that combine inference rules with a constantly evolving measure of time (a “now”) that itself can be referenced in those rules. At each time step, all possible one-step inferences are drawn by applying inference rules once to the present (working memory) knowledge, and marked with a timestamp. Allowing inferences only to be made based on one-step inferences on present knowledge, and not made by applying inference rules iteratively until the next time step, helps mitigate the “omniscience problem”, where all implications are treated as derived at once. By explicitly situating reasoning in time this way, contradictions can be dealt with as they arise in the inference process. In our example, Robbie encounters various contradictions, such as believing the room number is 128 but seeing it to be 123.

Time steps also aid in reasoning about past reasoning, and in the derivation of future theorems. Robbie for example would use the present value of “Now” in determining if there is sufficient time for her to complete her task (see Brody, Cox, & Perlis, 2013 for some details). Active logic differs from other temporal logics which lack a “now” represented as a changing time value; these other logics simply discriminate between a fixed past, present, and future.

Active logic is a non-monotonic reasoning scheme, meaning that inferences made in the past can be rejected and replaced with better ones in the present. The tagging of the reasoning process with time stamps allows the use of a belief’s history of acceptance/rejection during present reasoning; when direct contradictions arise in the knowledge base, this information can be of use. In particular, conflicts between expectation and observation can be recognized and reasoned about.

Each active logic belief is tagged with a unique identifier; this allows the reasoning mechanism to refer to inferences or assign properties to them – for instance, a belief can be distrusted, removed or assigned/reassigned a higher/lower priority. Thus, active logic is a natural mechanism for default reasoning (Purang, 2001) as well as resource-bounded reasoning and meta-reasoning (Josyula & M’Bale, 2013).

4. The Metacognitive Loop (MCL)

In our example, Robbie experienced several unexpected problems in the course of carrying out her task. To continue on mission, she needed to not only identify that there was a problem, but also implement a suitable response. We call such a reasoned anomaly-handling capability *generalized metacognition*. Ideally such a process can be largely domain-independent, involve only a modest amount of background knowledge and computation, and be implemented for any automated system. Much of our recent work has been aimed at testing this idea (e.g., see Schmill, Anderson, Fults, Josyula, Oates, Perlis, Haidarian, & Wilson, 2011). It essentially consists of three steps (the NAG-cycle): (i) monitor expectations to *note* any anomaly that might arise, (ii) *assess* it in terms of available responses, and (iii) *guide* any chosen response into place (and monitor the progress of that response for further anomalies). This requires, of course, expectations as to how things “ought” to be in the system, responses that could apply across the board to almost any type of anomaly encountered, and the ability to re-configure expectations in light of how things go. We refer to an algorithmic version of the NAG-cycle as the Metacognitive Loop (MCL) (Anderson & Perlis, 2005).

Our current generalized MCL module implements three special sets of abstract ontologies: an indications ontology for anomaly types, a failures ontology for assessment, and a responses ontology for repairs. The core of each ontology is currently implemented as a Bayesian network. These core nodes represent abstract and domain-general concepts of anomalies and how to respond to them. These nodes are linked within each ontology to express relationships between the concepts they represent. They are also linked between ontologies, allowing MCL to employ a number of Bayesian algorithms for reasoning over ontologies. Attached to the indications and responses ontologies are concrete “fringe” nodes. The fringe nodes for the indications core represent concrete, specific information about a possible expectation violation, and those for the responses core represent specific correction information. The host provides updates (e.g., sensor data) to the expectations fringe, and receives suggestions for repairs via the corrections fringe. When the expectation fringe nodes receive an update from the host, if the observed values in the update are different from the expected value specified in the fringe node, then an expectation violation occurs. The expectation violation triggers nodes in the indication ontology that correspond to the violation; the node activations propagate from the lower level, more specific fringe nodes to the higher level, more abstract indications of failures. The activations also get propagated to the failure ontology through the indications-failures inter-ontology links and to the response ontology through the failures-responses inter-ontology links. As the node activations propagate down the responses ontology to the more specific correction fringes, the correction fringe with the highest utility sends a specific correction to the host to act on.

Figure 1 depicts the host (shaded in yellow), and the generalized metacognition module MCL (shaded in blue). During operation, the host can also adjust or specify new expectations based on its ongoing experience. At the input interface, expectations are directly linked to the indications ontology through indication fringe nodes. At the output interface, the responses ontology’s fringe nodes are linked to a set of possible corrections that the host could employ. When an actual perturbation occurs in the host, MCL will detect the expectation violation through the input fringe nodes. It will then attempt to map it into the MCL core so that it may reason about it abstractly. MCL’s reasoning process then produces an output, which is articulated through the output fringe nodes in the form of an action that the host is able to carry out.

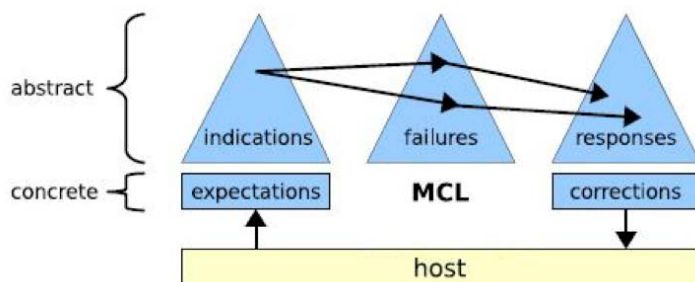


Figure 1. Metacognitive loop (MCL)

Returning to the example, Robbie encounters her first unexpected challenge as she enters the 10m stretch of cluttered hallway. She is under a time constraint to return the book. A cluttered hallway will force her to move slowly and may result in task failure. MCL can handle this situation in a number of ways. Robbie may have begun with outdated information and could not have realized this hallway was cluttered before reaching it; perhaps she would have chosen a different path had she known. Realizing the clutter would slow her down but also having already traveled to this stretch of hallway, Robbie estimates how long it should take her to move through the hallway and compares it with estimates for alternate routes. From her current location, travel via the cluttered hallway, with time added for the clutter, still presents the shortest estimated time to her goal and she decides to continue as planned. However, Robbie soon realizes that she is taking far longer than expected to move through the cluttered hallway. Sensing another expectation violation and reasoning about its cause and possible responses, Robbie decides to take another route. Alternatively, Robbie may not have a priori understood the negative relationship between clutter and travel time and started only with an expectation of the amount of time needed to move through a 10m stretch of hallway. When this expectation was violated, Robbie notes that something is wrong and reasons that there is something wrong with her current route, resolves to find a new route, and notes the relationship between clutter and travel time.

Continuing with the example, when Robbie reaches what she believes to be her destination, she may reasonably expect the room number written on the door to match the room number of her destination. When she reads '123' instead of '128', MCL would note an expectation violation and begin initiating behaviors to resolve the violation (in this case gathering further information by checking the two adjacent doors). In facing her third obstacle, Robbie notes a difficulty in opening the door and after initiating several behaviors aimed at resolving the problem, eventually concludes that she does not have the necessary abilities to achieve her goal and asks a human handler for help. Similarly, after using the key to open the door, Robbie realizes that the anticipated time to find the book will result in completion after the specified deadline. Having no known ability that would allow her to complete the task any faster, she returns to discuss the situation with a human.

5. NLP Dialog: ALFRED

Interactions of Robbie with humans can follow one of two design options: either train the humans to use robot-friendly commands, or train Robbie to use natural language. While the former is certainly feasible for experts in the robot's language, the latter is desirable for ease of use and

maximizing potential sources of information. Successful dialog management is also heavily reliant on metacognition (Anderson & Lee, 2005; McRoy, 1993) as well as learning (Rieger, 1974), in that general learning strategies can also apply to resolving anomalies in conversation. For instance, Robbie may not have understood the task specification, perhaps because of an unusual title (e.g., Smullyan's 1978 book entitled *What is the name of this book?*) or perhaps because we mumbled. Or, we may have summoned Robbie but fail to notice she has arrived, while she stands waiting for instructions. All of these scenarios will require some sort of dialog-specific reasoning strategies, which allow us to specify expectations and recovery strategies similar to those used in general reasoning. Towards this end, we have been working on a dialog agent named ALFRED (Active Logic For Reason-Enhanced Dialog).

ALFRED is a dialog agent which acts as an interface between a human user and a task-oriented domain (Josyula, 2005). It accepts English sentences as input and parses them into appropriate commands, based on the particular domain and information in its knowledge base (KB). ALFRED is designed to be a general agent and flexible enough to handle a variety of different scenarios. For each domain, ALFRED has a dictionary listing the possible commands and objects, as well as specifying the command syntax for that domain. To implement the NAG-cycle, ALFRED maintains a set of expectations regarding content, time and feedback. That is, it tracks what predicates are expected, when those predicates are expected, and the expected values of parameters in each predicate.

When an expectation is not met, ALFRED interprets it as an indication of an anomaly: noting the problem, assessing the situation, and guiding a response strategy into place. Some examples of responses might be to ignore, try again, adjust the plan or expectation, or ask for help. When the expectation is met, the corresponding violation is removed from the knowledge base; when it is not, ALFRED will attempt another response strategy until the issue is resolved or until it chooses *ignoring the violation* as the response strategy. We are currently conducting experiments that will compare the performance of two quite different implementations of MCL on a variety of different types of anomalies within ALFRED (McNany, Josyula, Cox, Paisner, Perlis, 2013). The two implementations differ in terms of how much of the host KB is shared by MCL. In one setup, generalized MCL is used as a monitor and control mechanism that runs external to the cognitive sphere of the host agent and hence any knowledge sharing between Alfred and generalized MCL is done explicitly. In the other experimental setup, a specialized MCL runs alongside the cognitive reasoner within the same active logic engine (and hence MCL has full access to the KB) to monitor and control the cognitive behaviors of the agent.

An example of metacognition within dialog involves using ALFRED as an interface to direct trains. In this setting, we have implemented the monitoring of the success of initiated responses and evaluation of candidate options before immediately initiating the same response again. For instance, if a user requests “send the Chicago train to New York”, ALFRED may choose Metroliner as the candidate, a train that is currently in Chicago. However, if the user replies “No” and repeats the same request, ALFRED evaluates its options, notices that its previous first choice of Metroliner was an unsuccessful response, and instead chooses Northstar, a train that originates in Chicago. In this way ALFRED is able to learn which entity is meant by “the Chicago train” instead of repeatedly choosing the same, incorrect train as a response to the user’s request.

6. Reinforcement Learning

Reinforcement Learning (RL) allows a robot to learn from an unknown environment. As mentioned in the running example, at first, Robbie knew very little. Therefore, all of her acquired knowledge about the world and herself are obtained via learning in one form or another. For instance, Robbie has previously learned a floor plan that she now consults to plan a path; Robbie tries the knob and cannot manage to turn it as she has been trained only on door handles; a knob is too hard for her grippers to manage, but if she uses a key, the door will open without having to turn the knob at all. That is, Robbie needs to learn at least to plan a path, to turn a handle, and to use a key to turn a knob. In the remainder of this section we illustrate one particular way that MCL can impact RL methods.

Consider a simple experiment with a standard RL algorithm (Q-learning; see Anderson & Perlis, 2005 for details). An agent is envisioned to maneuver within an 8x8 grid world, in which there is a positive reward in one corner, and negative in the opposite corner. The learner executes 10,000 turns, learning a very effective policy for maximizing reward, as is standard for this sort of learning algorithm. Then we switched the rewards and let the learner continue as before, for an additional 10,000 turns. Not unexpectedly, (see Figure 2) performance degraded. But what is striking is that it recovered far more slowly than it had done in the first 10,000 turns. In effect, it needed to “unlearn” what it had learned before it could then learn the new reward structure.

This of course is not very intelligent. A smarter agent would soon realize that its well-learned strategy no longer worked at all, would stop using it, and would start running the reinforcement over from scratch. When we configured the agent to do that, it learned the new reward structure much faster (see Figure 3 where both old and new methods are superimposed).

We anticipate that MCL can similarly be used to enhance many kinds of RL and other system components.

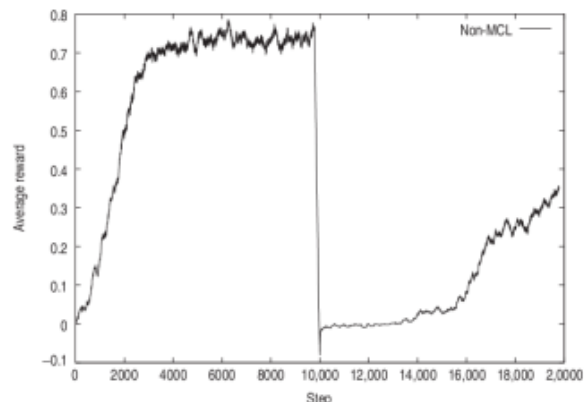


Figure 2. Performance of Q-learner, over 20,000 turns, with initial reward structure of [-10 10] and a post-perturbation reward structure of [10 -10]

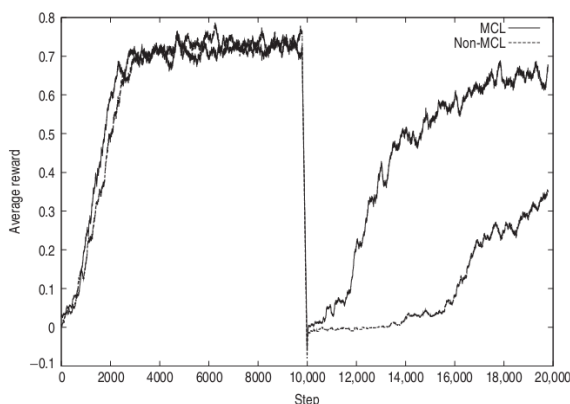


Figure 3. Performance of same Q-learner but with MCL, superimposed on Figure 2

7. Goal-Driven Autonomy and an Integrated Metacognitive Architecture

As Robbie leaves our office, it dawns on her that the failure of delivering the book may reoccur the next time she is asked for another book and that even humans may have difficulty finding books they want if they are out of order. Given that she has nothing to do at the moment, she decides to clean up the library and reorganize the book shelves. She remembers being scolded once for borrowing a book without asking first, so she returns to ask our permission. But we are not there, having gone to a noon appointment. Robbie reflects and then decides that rearranging is not the same as borrowing, and that it would be ok to organize the books.

Autonomy has long been viewed as effectively performing tasks to automatically achieve the goals given to an agent by a human and learning to improve such performance in the future. But a new model of agent autonomy called *goal-driven autonomy* (GDA) asserts that autonomy is also about recognizing novel problems, explaining what caused such problems, and generating one's own goals to solve the problems (Cox, 2007; 2013; Klenk, Molineaux, & Aha, 2013; Paisner, Maynard, Cox, & Perlis, in press). As such this is a variation of the note-assess-guide procedure.

A GDA agent notes when failures occur (Robbie sought to achieve her goal but did not succeed), assesses the failure (Robbie failed because the book was not shelved correctly), and then guides a response to the failure (Robbie generated a goal to correctly shelve the books). Here the note phase is similar: an observation does not match the expectation and hence a discrepancy (i.e., anomaly) exists. However the assess phase involves determining a causal explanation for the failure or discrepancy. The response is to generate a new goal to solve the problem. The generation of such goals can be found by determining a salient antecedent of the explanation and negating it (Cox, 2007; 2013). Here the robot generates the goal of not having the books being shelved incorrectly.

We have been working on implementing a larger cognitive architecture that integrates much of this work. The *Metacognitive, Integrated, Dual-Cycle Architecture (MIDCA)* (Cox, Maynard, Paisner, Perlis, & Oates, 2013; Cox, Oates, & Perlis, 2011; Maynard, Cox, Paisner, & Perlis, 2013) consists of action-perception cycles at both the cognitive (i.e., object) level and the metacognitive (i.e., meta-) level. The output side of each cycle consists of intention formation, planning, and action execution, whereas the input side consists of perception, interpretation, and goal evaluation. A cycle selects a goal and commits to achieving it. The agent then creates a plan to achieve the goal and subsequently executes the planned actions to make the world match the goal state. The agent perceives changes to the environment resulting from the actions, interprets the percepts with respect to the plan, and evaluates the interpretation with respect to the goal. At the object level, the cycle achieves goals that change the environment. At the meta-level, the cycle achieves goals that change the object level. That is, the metacognitive perception components introspectively monitor the processes and mental state changes at the cognitive level. The action component consists of a meta-level controller that mediates reasoning over an abstract representation of the object-level cognition.

To illustrate these distinctions, consider the object level as shown in Figure 4. Here the meta-level executive function manages the goal set \mathcal{G} . In this capacity, the meta-level can add initial goals (g_0), subgoals (g_s) or new goals (g_n) to the set, can change goal priorities, or can change a particular goal (Δg). In problem solving, the Intend component commits to a current goal (g_c) from those available by creating an intention to perform some task that can achieve the goal. The Plan component then generates a sequence of actions (π_k , e.g., a hierarchical-goal-net plan Shivashankar, Kuter, Nau, & Alford, 2012; Shivashankar, Alford, Kuter, & Nau, 2013) that instantiates that task

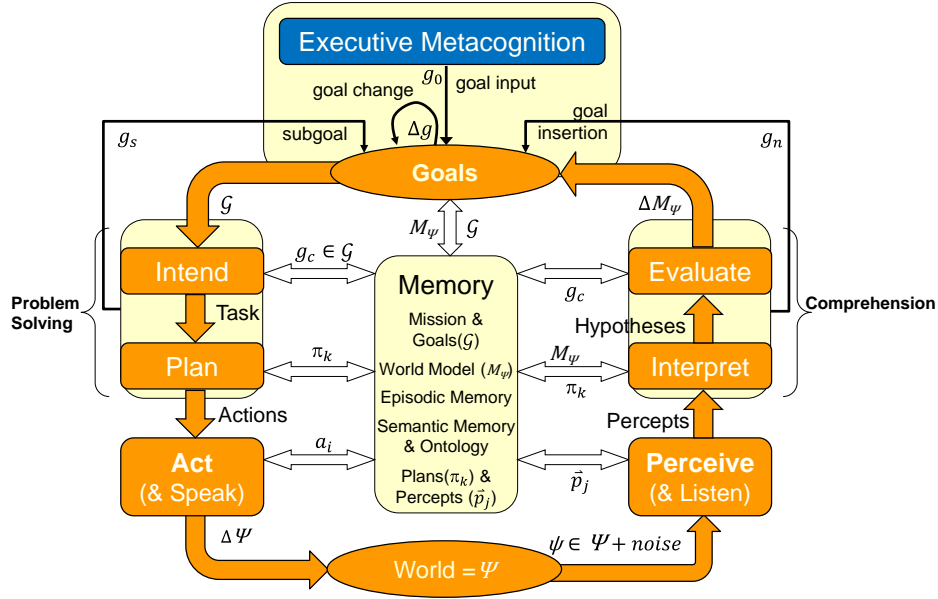


Figure 4. Object-level detail with meta-level goal management

given the current model of the world (M_ψ) and its background knowledge (e.g., semantic memory and ontologies). The plan is executed to change the actual world (Ψ) through the effects of the planned actions (a_i). The goal and plan are stored in memory and constitute the agent's expectations about how the world will change in the future. Then given these expectations, the comprehension task is to understand the execution of the plan and its interaction with the world with respect to the goal.

Comprehension starts with perception of the world in the attentional field. Interpretation takes as input the resulting percepts (\vec{p}_j) and the expectations in memory (π_k and g_c) to determine whether the agent is making sufficient progress. An MCL note-assess-guide procedure implements the comprehension process. The procedure is to note whether an anomaly has occurred; assess potential causes of the anomaly by generating hypotheses; and guide the system through a response. Responses can take various forms, such as (1) test a hypothesis; (2) ignore and try again; (3) ask for help; or (4) insert another goal (g_n). In the absence of an anomaly, the agent incorporates the changes inferred from the percepts into the world model (ΔM_ψ) and the cycle continues. This cycle of problem-solving and action followed by perception and comprehension, functions over discrete state and event representations of the environment.

8. Memory

Memory is a crucial function for a perpetual cognitive agent. Conceptual information in a *semantic memory* is functionally important for interpreting perceptions and for reasoning about the world. But equally important, an *episodic memory* stores a personal history of the agent and its interactions

with the world and other agents in that world (e.g., see Laird, Nuxoll, & Derbinsky 2012). If the agent is to reason about its capacity to perform actions in the present, it is important that it knows what worked and did not work in the past. It is not always possible to infer results, especially when it comes to the behavior of humans; so for instance, remembering the likes and dislikes of others (e.g., remembering that asking permission is important to someone) is a useful function. Furthermore, rather than having to solve problems from scratch each time, an agent should remember how it solved similar problems in the past and simply reuse the past solution or adapt an old solution to fit new circumstances. Such a case-based reasoning approach (de Mántaras, et al., 2006; Kolodner, 1993) has been shown to reduce effort and make for efficient problem solving (e.g., Cox, Munoz-Avila, & Bergmann, 2006; Veloso, 1994).

Memory should organize conceptual and procedural information in a manner that makes it effective. A good memory retains useful information and makes it available at the right time in the right form (Schank, 1982). Memory in cognitive agents can be partitioned into separate functions and controlled by an inference cycle mechanism; e.g., see (Elgot-Drapkin, Miller, & Perlis, 1991) in which preliminary experiments illustrated a critical impact of the size of working memory/STM. The benefit of a good memory architecture is that knowledge need not be searched by an arbitrary brute-force approach; rather an agent can depend upon a retrieval match between a contextual cue and the index used to store a memory. The cost is in terms of what has been called the indexing problem (Kolodner, 1993; Schank, 1982; Schank & Osgood, 1990). The problem is to choose effective cues, or features in an input, to be used as probes for retrieving from memory the knowledge structures necessary to process an input.

The converse problem is the problem of forgetting (Cox, 1994). If the cues are not chosen with care during retrieval time, or if the indexes are not chosen well during encoding, the reasoner may not recall a memory structure when it is needed. The forgetting problem is to reorganize memory and the indexes by which memory is accessed. Because reasoning failures may occur due to faulty memory organization, as well as because of faulty reasoning components or faulty knowledge, the selection or retrieval of knowledge plays an important role in determining the cause of failure.

Reasoning failures related to information retrieval have been addressed using metamemory (Caro, Jimenez, & Paternina, 2012; Leake, 1995). In artificial intelligence, metamemory refers to the processes and techniques a system uses to monitor and control its own memory, which has strong parallels in cognitive psychology research (Nelson, Narens, & Dunlosky, 2004; Metcalfe & Dunlosky, 2008). Indeed, to realize one's own memory limitations – no memory architecture will be perfect in retrieving exactly the right information at the optimal moment – is an important piece of self-knowledge that can guide an agent's behavior (e.g., setting reminders for itself). Keeping all information – even just all episodic information – in working memory where the agent's reasoning processes can run rampant on it (the swamping or omniscience problem) will only clog the agent's ability to act in a timely way. So metamemory can have a role in helping an agent mark items that are important to retain for a time (Robbie marks the book-title information that way) and allow others to be gracefully “forgotten.”

9. Related Work

Since McCarthy originally described the concept of a computer advice taker (McCarthy, 1959), many research projects have embraced the goal of implementing persistent agents that co-exist and interact with humans over extended time periods. The original work at SRI on Shakey the robot

(Nilsson, 1984) combined sub-systems that reasoned using logic and that acted through sensors and effectors on the platform. It represented the first effort to build a human-like intelligent physical and mental system (i.e., an instantiation of sorts of Nilsson's, 1983, *computer individual*), although it was extremely brittle given any unforeseen circumstances.

Our idea of a perpetual cognitive real-world agent relates to variations on autobiographical agents that have a memory of their own experiences (e.g., Dautenhahn, 1998; Derbinsky & Laird, 2010), social agents that interact and cooperate with humans and other agents (e.g., Breazeal, & Scassellati, 1999; Scassellati, 2001), and developmental cognitive robots that learn over time (e.g., Weng et al., 2001). Researchers have approached the research in various ways resulting in theories of human-level intelligence (Cassimatis, 2012; Cassimatis & Winston, 2004) and artificial general intelligence (Wang & Goertzel, 2012; Crowder & Friess, 2010).

A number of more recent research projects exist within the artificial intelligence and cognitive science communities that integrate multiple high-level cognitive functions and perform complex tasks in dynamic environments, some with actual physical platforms or robots. Well known examples include ACT-R (Anderson & Lebiere, 1998), CogAff (Sloman, 2003, 2011), Companion Cognitive Systems (Forbus, Klenk, & Hinrichs, 2009), EM-One (Singh, 2005), DIARC (Krause, Schermerhorn, & Scheutz, 2012), EPILOG (Morbin & Schubert, 2011), Icarus (Langley & Choi, 2006), SALS (Morgan, 2009), SNePS (Shapiro, 2000), Soar (Laird, 2012), and SS-RICS (Kelley, 2003).

10. Current and Future Plans and Conclusion

We are currently working on a variety of additional aspects of our long-term goal of human-level autonomous systems. One large portion of the work involves combining Alfred/Active Logic with MCL; as a particular example, we are investigating metacognitive means to allow a conversational agent to deal with unanticipated pauses in a conversation (McNany, Josyula, Cox, Paisner, & Perlis, 2013). With regard to reinforcement learning, we are exploring the use of the natural-actor critic algorithm in conjunction with the growing neural gas (GNG) algorithm to help a robot learn the effects of its actions, like a baby thrashing about until it learns that motor impulses and visual and tactile inputs correlate in highly regular ways. And with regard to MIDCA itself, we are using a symbolic version of the A-distance algorithm along with GNG to help a system identify (note) anomalies (Paisner, Perlis, & Cox, in press).

We believe that a frontal assault on the challenge of human-level AI is timely, that many of the needed tools are currently available, and that many of the very real remaining gaps can be filled along the lines we have sketched here. One particular thrust that we envision is that of a competitive robot treasure-hunt with, say, PR2 robots (that have a considerable degree of fine manipulative capacity as well as ease of programming). The treasure-hunt domain nicely combines natural-language, perception, real-time planning, goal-creation, action, and indeed most of AI.

Acknowledgements

This material is based upon work supported by ONR Grants # N00014-12-1-0430 and # N00014-12-1-0172 and by ARO Grant # W911NF-12-1-0471.

References

- Anderson, J. R., & Lebiere, C. (1998). *The atomic components of thought*. Mahwah, N. J: LEA, 1998.
- Anderson, M. L., Josyula, D., Okamoto, Y., & Perlis, D. (2002). Time-situated agency: Active logic and intention formation. In *Proceedings of the 25th German Conference on Artificial Intelligence*.
- Anderson, M. L., & Perlis, D. (2005). Logic, self-awareness and self-improvement: The metacognitive loop and the problem of brittleness. *Journal of Logic and Computation*, 15(1).
- Anderson, M. L., & Lee, B. (2005). Metalanguage for dialog management. I. In *Proceedings of the 16th Annual Winter Conference on Discourse, Text and Cognition*.
- Breazeal, C., & Scassellati, B. (1999). How to build robots that make friends and influence people. *1999 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS-99)*. Kyongju, Korea.
- Brody, J., Cox, M. T., & Perlis, D. (2013). The processual self as cognitive unifier. In *Proceedings of the Annual Meeting of the International Association for Computing and Philosophy*. IACAP-2013.
- Caro, M., Jimenez, J., & Paternina, A. (2012). Architectural modeling of metamemory judgment in case-based reasoning systems. *2012 XXXVIII Conferencia Latinoamericana En Informatica (CLEI)* (pp. 1–8).
- Cassimatis, N. L. (2012). Human-level artificial intelligence must be an extraordinary science *Advances in Cognitive Systems 1*, 37- 45.
- Cassimatis, N., & Winston, P. (Eds.) (2004). *Achieving human-level intelligence through integrated systems and research: Papers from the AAI fall symposium*. Technical Report FS-04-01. Palo Alto, CA: AAI Press.
- Cox, M. T. (1994). Machines that forget: Learning from retrieval failure of mis-indexed explanations. In A. Ram & K. Eiselt (Eds.), *Proceedings of the Sixteenth Annual Conference of the Cognitive Science Society* (pp. 225-230). Hillsdale, NJ: Lawrence Erlbaum Associates.
- Cox, M. T. (2007). Perpetual self-aware cognitive agents. *AI Magazine* 28(1), 32-45.
- Cox, M. T. (2013). *Goal-driven autonomy and question-based problem recognition*. Submitted.
- Cox, M. T., Maynard, M., Oates, T., Paisner, M., & Perlis, D. (2013). The integration of cognitive and metacognitive processes with data-driven and knowledge-rich structures. In *Proceedings of the Annual Meeting of the International Association for Computing and Philosophy*. IACAP-2013.
- Cox, M. T., Munoz-Avila, H., & Bergmann, R. (2006). Case-based planning. *Knowledge Engineering Review* 20(3), 283-287.
- Cox, M. T., Oates, T., & Perlis, D. (2011). Toward an integrated metacognitive architecture. In P. Langley (Ed.), *Advances in Cognitive Systems, papers from the 2011 AAI Symposium* (pp. 74-81). Technical Report FS-11-01. Menlo Park, CA: AAI Press.
- Crowder, J. A., & Friess, S. (2010). Metacognition and metamemory concepts for AI systems. In *International Conference on Artificial Intelligence, ICAI 2010*. Las Vegas.

- Dautenhahn, K. (1998). Meaning and embodiment in life-like agents. In C. Nehaniv (Ed.), *Plenary working papers in computation for metaphors, analogy and agents* (pp. 24-33). University of Aizu Technical Report 98-1-005.
- De Mántaras, R. L., McSherry, D., Bridge, D., Leake, D., Smyth, B., Craw, S., Faltings, B., Maher, M. L., Cox, M. T., Forbus, K., Keane, M., Aamodt, A., & Watson, I. (2006). Retrieval, reuse and retention in case-based reasoning. *Knowledge Engineering Review*. 20(3), 215-240.
- Derbinsky, N., & Laird, J. E. (2010). Extending soar with dissociated symbolic memories. In M. Lim & W. Ho (Eds.), *Proceedings of the Remembering Who We Are: Human Memory for Artificial Agents Symposium*, At the AISB 2010 convention, De Montfort Univ., Leicester, UK.
- Elgot-Drapkin, J., Miller, M., & Perlis, D. (1991). Memory, reason, and time: The step-logic approach. In Cummins and Pollock (Eds.), *Philosophy and AI: Essays at the interface*. Cambridge, MA: MIT Press.
- Forbus, K., Klenk, M., & Hinrichs, T. (2009). Companion cognitive systems: Design goals and lessons learned so far. *IEEE Intelligent Systems*, 24, 36–46.
- Josyula, D. (2005). *A unified theory of acting and agency for a universal interfacing agent*, Ph.D. dissertation, University of Maryland, College Park.
- Josyula, D and M'Bale, K. (2013). Bounded Metacognition. In *Proceedings of the Fifth International Conference on Advanced Cognitive Technologies and Applications* (pp.147-152). Red Hook, NY: Curran Associates.
- Kelley, T. (2003). Symbolic and sub-symbolic representations in computational models of human cognition: What can be learned from biology? *Theory and Psychology*, 13(6), Dec. 2003.
- Klenk, M., Molineaux, M., & Aha, D. (2013) Goal-driven autonomy for responding to unexpected events in strategy simulations. *Computational Intelligence* 29(2), 187–206.
- Kolodner, J. L. (1993). *Case-based reasoning*. San Mateo, CA: Morgan Kaufmann Publishers.
- Krause, E., Schermerhorn, P., & Scheutz, M. (2012). Crossing boundaries: Multi-level introspection in a complex robotic architecture for automatic performance improvements. In *Proceedings of the Twenty-Sixth Conference on Artificial Intelligence*. Palo Alto, CA: AAAI Press.
- Laird, J. E. (2012). *The Soar Cognitive architecture*. Cambridge, MA: MIT Press.
- Laird, J. E., Nuxoll, A., & Derbinsky, N. (2012). Episodic memory. In J. Laird, *The Soar cognitive architecture* (pp. 225-246). Cambridge, MA: MIT Press.
- Langley, P. (2012). The cognitive systems paradigm. *Advances in Cognitive Systems* 1, 3–13.
- Langley, P., & Choi, D. (2006). A unified cognitive architecture for physical agents. *The Twenty-First National Conference on Artificial Intelligence*. Menlo Park, CA: AAAI Press.
- Leake, D. B. (1995). Representing Self-knowledge for Introspection about Memory Search A Planful Framework for Internal Reasoning. In M. T. Cox & M. Freed (Eds.), *AAAI Spring Symposium on Representing Mental States and Mechanisms*. Stanford, CA.
- Maynard, M., Cox, M. T., Paisner, M., & Perlis, D. (2013). Data-driven goal generation for integrated cognitive systems. In C. Lebiere & P. S. Rosenbloom (Eds.), *Integrated Cognition: Papers from the AAAI Fall Symposium* (pp. 47-54). Menlo Park, CA: AAAI Press.
- McCarthy, J. (1959). Programs with common sense. In *Symposium Proceedings on Mechanisation of Thought Processes* (Vol. 1, pp. 77-84). London: Her Majesty's Stationary Office.

- McNany, E., Josyula, D., Cox, M. T., Paisner, M., Perlis, D. (2013). Metacognitive guidance in a dialog agent. In *Proceedings of the Fifth International Conference on Advanced Cognitive Technologies and Applications* (pp.137-140). Red Hook, NY: Curran Associates.
- McRoy, S. (1993). *Abductive interpretation and reinterpretation of natural language utterances*. Ph.D. dissertation, University of Toronto, 1993.
- Metcalfe, J., & Dunlosky, J. (2008). Metamemory. In H. L. Roediger III (Ed.), *Cognitive Psychology of Memory. Vol. 2 of Learning and Memory: A comprehensive reference*, 4 vols.
- Morbini, F., & Schubert, L. (2011). Metareasoning as an integral part of commonsense and autocognitive reasoning. In M. T. Cox & A. Raja (Eds.) *Metareasoning: Thinking about thinking* (pp. 267-282). Cambridge, MA: MIT Press.
- Morgan, B. (2009). Funk2: A distributed processing language for reflective tracing of a large critic-selector cognitive architecture. In *Proceedings of the Metacognition Workshop at the Third IEEE International Conference on Self-Adaptive and Self-Organizing Systems*. San Francisco, CA.
- Nelson, T., Narens, L., & Dunlosky, J. (2004). A revised methodology for research on metamemory: Pre-judgment recall and monitoring (PRAM). *Psychological methods*, 9(1), 53–69.
- Nilsson, N. (1983). Artificial intelligence prepares for 2001. *AI Magazine* 4(4), 7-14.
- Nilsson, N. (1984). *Shakey the robot*. Technical Note 323, Artificial Intelligence Center, Menlo Park, CA: SRI.
- Paisner, M., Maynard, M., Cox, M. T., & Perlis, D. (in press). Goal-driven autonomy in dynamic environments. To appear in D. W. Aha, M. T. Cox, & H. Munoz-Avila (Eds.), *Proceedings of the 2013 Annual Conference on Advances in Cognitive Systems: Workshop on Goal Reasoning* (Tech. Rep. No. CS-TR-5029). College Park, MD: University of Maryland, Department of Computer Science.
- Paisner, M., Perlis, D., & Cox, M. T. (in press). Symbolic anomaly detection and assessment using growing neural gas. To appear in *Proceedings of the 25th IEEE International Conference on Tools with Artificial Intelligence*. Los Alamitos, CA: IEEE Computer Society.
- Purang, K. (2001) Alma/Carne: Implementation of a Time-Situated Meta-Reasoner. In *Proceedings of the International Conference on Tools with Artificial Intelligence*, 103-110.
- Rieger, C. (1974). *Conceptual memory: A theory and computer program for processing the meaning content of natural-language utterances*. Ph.D. dissertation, Stanford University, 1974.
- Scassellati, B. M. (2001). *Foundations for a theory of mind for a humanoid robot*. Doctoral dissertation, Massachusetts Institute of Technology, Cambridge.
- Schank, R. C. (1982). *Dynamic memory: A theory of reminding and learning in computers and people*. Cambridge, MA: Cambridge University Press.
- Schank, R. C., & Osgood, R. (1990). *A content theory of memory indexing*. Technical Report 2. Institute for the Learning Sciences, Northwestern University, Evanston, IL.
- Shamwell, J., Oates, T., Bhargava, P., Cox, M. T., Oh, U., Paisner, M., & Perlis, D. (2012). The robot baby and massive metacognition: Early steps via growing neural gas. In *Proceedings of the IEEE Conference on Development and Learning - Epigenetic Robotics 2012*. Los Alamitos, CA: IEEE.

- Shapiro, S. (2000). SNePS: A logic for natural language understanding and commonsense reasoning. In L. Iwanska & S. Shapiro (Eds.), *Natural language processing and knowledge representation: Language for knowledge and knowledge for language* (pp. 175–195). Menlo Park, CA: AAAI Press/The MIT Press.
- Shivashankar, V., Alford, R., Kuter, U., & Nau, D. (2013). The GoDeL planning system: A more perfect union of domain-independent and hierarchical planning. In *Twenty-Third International Joint Conference on Artificial Intelligence (IJCAI-13)*.
- Shivashankar, V., Kuter, U., Nau, D., & Alford, R. (2012). A hierarchical goal-based formalism and algorithm for single-agent planning. In V. Conitzer, M. Winikoff, L. Padgham, & W. van der Hoek (Eds.), *Proceedings of the Eleventh International Conference on Autonomous Agents and Multiagent Systems* (pp. 2380-2386). International Foundation for Autonomous Agents and Multiagent Systems.
- Schmill, M., Anderson, M., Fults, S., Josyula, D., Oates, T., Perlis, D., Haidarian, S., & Wilson, S. (2011). The metacognitive loop and reasoning about mistakes. In M. T. Cox & A. Raja (Eds.), *Metareasoning: Thinking about thinking* (pp. 183-198). Cambridge, MA: MIT Press.
- Singh, P. (2005). *EM-ONE: An architecture for reflective commonsense thinking*. Ph.D. dissertation. Department of Electrical Engineering and Computer Science. MIT. Boston, MA.
- Sloman, A. (2003). *The Cognition and Affect project: Architectures, architecture-schemas, and the new science of mind* (Technical Report). Birmingham, UK, School of Computer Science, University of Birmingham.
- Sloman, A. (2011). Varieties of meta-cognition in natural and artificial systems. In M. T. Cox & A. Raja (Eds.), *Metareasoning: Thinking about thinking* (pp. 307–323). Cambridge, MA: MIT Press.
- Smullyan, R. M. (1978). *What is the name of this book? The riddle of Dracula and other logical puzzles - knights, knaves, and other logic puzzles*. New York: Touchstone Books.
- Veloso, M. (1994). *Planning and learning by analogical reasoning*. Berlin: Springer-Verlag.
- Wang, P., & Goertzel, B. (2012). *Theoretical foundations of artificial general intelligence*. Berlin: Springer.
- Weng, J., McClelland, J., Pentland, A., Sporns, O., Stockman, I., Sur, M., & Thelen, E. (2001). Autonomous mental development by robots and animals. *Science* 291, 599-600.

Metamemory for Information Retrieval from Long-term Memory in Artificial Cognitive Systems

Manuel F. Caro

Department of Educational Informatics, University of Córdoba, Montería – Colombia

MFCAROP@UNAL.EDU.CO

Darsana P. Josyula

Department of Computer Science Bowie State University, Bowie, MD – USA

DARSANA@CS.UMD.EDU

Jovani A. Jiménez

Department of Computer and Decision Science, National University of Colombia, Medellín - Colombia

JAJIMEN1@UNAL.EDU.CO

Abstract

In this paper we describe a novel rule-based architecture of metamemory called M²-Acch. M²-Acch involves the use of metamemory for adapting to changes in the constraints for information retrieval from long-term memory in artificial cognitive systems. M²-Acch is composed of a three layer structure: static layer, functional layer and information layer. The structural components of each layer model are described using formal definitions. M²-Acch uses confidence judgments for recommending search strategies for adaptation to changes in the information retrieval constraints. It was implemented and validated in an intelligent tutoring system named FUNPRO. The results of the experimental tests show that M²-Acch can be used as a valid tool for adapting to changes in the constraints in information retrieval from long-term memory in artificial cognitive systems.

1 Introduction

Retrieval is one of the basic processes of memory activities (Düzel et al., 1999; Metcalfe & Dunlosky, 2008; Schank, 1982) and has been studied for many years e.g., (Ebbinghaus, 1962). Retrieval includes tasks associated with accessing of stored information. One crucial influence on the outcome of any retrieval process is the knowledge available to that process (Leake, 1995). This knowledge includes search constraints (Kizilirmak, Rösler & Khader, 2012), parameters and other information related to the target of the search (Unsworth, 2010). The search constraints restrict the information retrieved by the search task, by influencing the search strategy used to fulfill the search task goal (Mecklinger, 2010).

Changes in search constraints affect the performance of information retrieval (Huet & Mariné, 1997; Kizilirmak et al., 2012). When there are changes in constraints, information retrieval cannot be done effectively by the same search strategy for all cases. Thus the system needs to assess changes in the constraints of the search tasks and select the most appropriate search strategy.

In the context described above, we propose the use of metamemory as a mechanism that allows adaptation to changes in the constraints of a search task. Metamemory is a component of metacognition (Cox, Oates, & Perlis, 2011). As such, it deals with the memory capabilities of a system and strategies that aid memory (Crowder & Friess, 2010; Flavell & Wellman, 1977), as well as the processes involved in self-monitoring of the memory. In our approach, adaptation is retrieved by first monitoring and identifying changes in the constraints of the search task; once a change in the constraints is detected, then the metamemory assesses the knowledge available

to satisfy the constraint and suggests the most appropriate search strategy. In order to monitor and identify changes in the constraints, the meta-level stores expectations about the object-level information like judgments, observations and memory events.

Sutcliffe and Ennis (1994) developed a framework to create explanatory and predictive theories of information searching to improve the design of information retrieval (IR) systems. The framework is developed into a cognitive theory of information searching by the addition of strategies and correspondence rules; however it focuses on predicting user behavior and not on self-monitoring of cognitive system memory. Leake (1995) propose a framework for modeling introspective reasoning. The framework has relevance for modeling introspective reasoning about memory search, but the framework does not take into account the restrictions on search tasks, and does not explain how a system could be adapted to changes in search constraints.

More recently several studies have been published about information retrieval, particularly related to recommendation systems. Gomes, Braga & Borges (2012) present a model where the memory processes are organized functionally in hierarchical levels such that higher levels coordinate sets of functions at a lower level; nevertheless this model does not take into account the constraints of information search tasks. Duch and Szymański (2007) present a web game based on a computational model of semantic memory that implements semantic search algorithms for information retrieval, ut is based in simplification of semantic memory. Develaar, Yu, Harbison, Hussey and Dougherty (2013) describe a set of rules for memory search termination that can be used to improve information retrieval performance, but do not implement metamemory, or take into account the restrictions on information search tasks. In our study of the literature to date, we have not found other computational work that uses metamemory as a mechanism for adaptation to changes in the constraints of the search task for information retrieval.

The paper is structured as follows. In Section 2, we describe the metamemory model and present an intelligent tutoring system that was used for validation. In this section several cases are described, where a search task is invoked with variations in the constraints. Section 3 then presents empirical studies of an implementation of M^2 -Acch in an intelligent tutoring system. Finally, we present the discussion and conclusions.

2 Metamemory model for adapting to constraint changes (M^2 -Acch)

M^2 -Acch (pronounced “match”) is a Metamemory Model for Adapting to Constraints Changes in the information retrieval tasks in cognitive systems. M^2 -Acch consists of a cycle of reasoning about events that occur in long-term memory (LTM). The reasoning cycle inputs are the memory events that occur in LTM and the output consists of recommendations which may vary according to the memory events. In particular we will focus on the reasoning process that allows adaptation to constraint changes related to retrieving information from LTM.

M^2 -Acch comprises a three-layer modeling. *The static layer* contains the structural model that describes the static relationships between the elements of the meta-memory architecture. *The dynamic layer* comprises the functional model that describes the processes of reasoning and decision making. Finally, *the information layer* describes the flow of information between the cognitive system and M^2 -Acch. These three layers are presented bellow in sections 2.1, 2.2 and 2.3.

2.1 Structural model (the static layer)

The structural model is composed of an ontology that comprises two types of elements: structural elements and functional elements. Structural elements are containers into which the functional elements are embedded; the main structural element is the *level*. The functional elements enable reasoning and decision-making, functional elements are: *event*, *strategy*, *goal*, *constraint*, *judgment*, *process*, *expectation* and *sensor*.

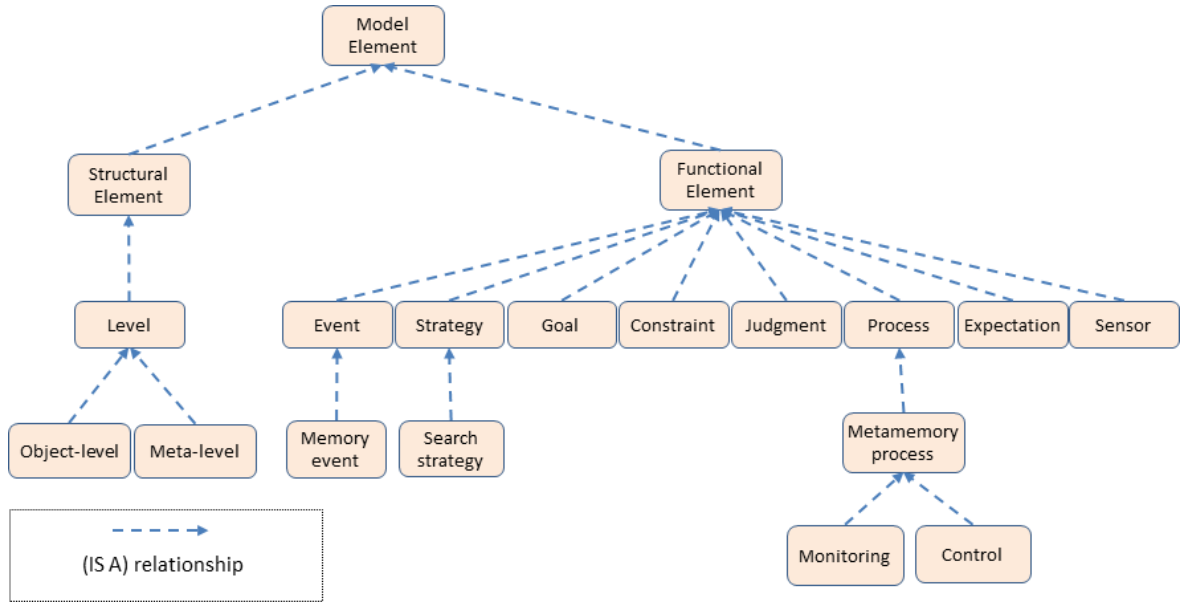


Figure 1. Structural model of M²-Acch

2.1.1 Description of components

In this section, we introduce the basic concepts of the formalism to describe the structural model of M²-Acch.

We will use an example to facilitate comprehension of the definitions. The example is based on an intelligent tutoring system that receives as input some student data and its output is the content recommendation. The recommendations are based on an instructional plan adapted to the student.

Definition 1. $T = \{L, E, G, D, J, S\}$ is the set containing the basic types of elements of the meta-memory model, where:

- L represents the structural levels of the model.
- E describes the events that occur in a system that directly affect LTM.
- G is the goal of an event.
- D represents the constraints required to perform a search.
- J represents the judgments that the meta-level triggers about events E that happens at the object-level.
- S is the search strategy recommended by the meta-level.

Definition 1.1. M²-Acch is composed of two main levels (L): *object-level* and *meta-level*. The object-level contains the LTM of the cognitive system and the meta-level contains *knowledge about LTM* and *processes for reasoning about LTM*.

Definition 1.2. The events (E) represent actions that are performed on the memory. $E = \{ID, y, g, d, t\}$ is the set of components that represents the structure of an event, where:

- ID is the unique identifier of the event.

y is the type of the event, $y \in Y$ and $Y = \{\text{call}, \text{execute}, \text{re-configure}\}$.
 g is the goal of the event.
 d is the constraint of the event.
 t is the memory task that originated the event.

The events that occur in the object-level can be of different types, for this particular research we work with three types of events: i) *call* if the event is a call to a search, acquisition or retention task on memory. This type of event is previous to the execution of the task. This event indicates to the system that a specific task on memory is required; ii) *execute* indicates that a search, acquisition or retention task is running on memory; iii) *re-configure* indicates that a search, acquisition or retention task has failed and it needs to be reconfigured.

Definition 1.3. Goals (G) are subcomponents of events. Each event can have only one goal. Goals contain relevant meta-data about information to be stored or retrieved from memory. $G = \{\text{ID}, a, t, s, r\}$ is the set of components that represents the structure of a goal, where:

ID is the unique identifier of the goal.
 a is an action performed on memory, $a \in A$ and $A = \{\text{acquisition}, \text{retention}, \text{retrieval}\}$.
 t is the target of the action a .
 s is the state of the goal, $s \in S$ and $S = \{\text{starting}, \text{waiting}, \text{working}, \text{finished}\}$.
 r represents the final result of the goal, $r \in R$ and $R = \{\text{satisfied}, \text{unsatisfied}\}$.

For illustration the intelligent tutoring system as example: if the system is doing a search of resources for a student's lesson then the type of event memory is *execute*; the goal action is *retrieval*; the goal status is *working* and the goal result will depend on the success or failure of the search.

Definition 1.4. The constraints (D), in an event (E) refer to the information requirements that must be satisfied so that the *event* fulfills the goals. $D = \{\text{ID}, K, X, Q, y\}$ is the set of components that represents the structure of a constraint, where:

ID is the unique identifier of the goal.
 $K = \{k_1, \dots, k_n\}$ represents the set of information requirements needed to retrieve or store the target of the goal.
 $X = \{x_1, \dots, x_n\}$ is the set of information excluded from retrieval.
 $Q = \{q_1, \dots, q_n\}$ is the set of special requirements needed to retrieve or store the target of the goal.
 y is the type of the constraint, $y \in Y$ and $Y = \{\text{basic}, \text{complex}\}$.

Using the example presented in definition 1.3, if the intelligent tutoring system recommends a resource bad evaluated by student, then the resource is excluded from a new search with similar settings.

Definition 1.5. Metacognitive judgments (J) represent assessments performed in the meta-level about events that occur in memory. These judgments provide information that the system uses to determine whether it is able to attempt retrieval or storage. In the current model, we have two types of metacognitive judgments, these are:

COP (Certainty of Optimal Performance) measures the degree of certainty that the system has with regarding to optimum performances obtained in the past, having constraints similar to the current user.

CSRD (Certainty of Satisfying the Retrieval constraints) measures the degree of certainty that the system has with regard to the level of knowledge that the system possesses to attend the requirements of the retrieval constraints.

Definition 1.6. The meta-level contains a schema with information search strategies (S) available at the object-level. A major meta-level function is to recommend the most appropriate search strategy for the constraints of information retrieval from LTM.

$S = \{s_1, \dots, s_n\}$ is the set of search strategies that are available in the object-level, with $S \neq \emptyset$. The number of available strategies depends on the particular implementation of each system.

Definition 1.7. The expectations are behaviors that the cognitive system is expecting to pass in object-level. Expectations are associated to sensors. The sensors are elements that the meta-level has to monitor information flows coming from the object-level. The information contained in the sensor is called observation. When observations do not match with expectations, then there is a violation of expectation. The violations of expectations are interpreted by the meta-level as failures in the reasoning of object-level.

2.2 Functional model (the dynamic layer)

This section describes the processes for monitor and control of LTM that have been incorporated into M^2 -Acch. Monitoring processes include mechanisms for detecting events in LTM and performing deep search processes on the meta-level knowledge about the object-level. Moreover, control processes include rules for the recommendation of search strategies on LTM.

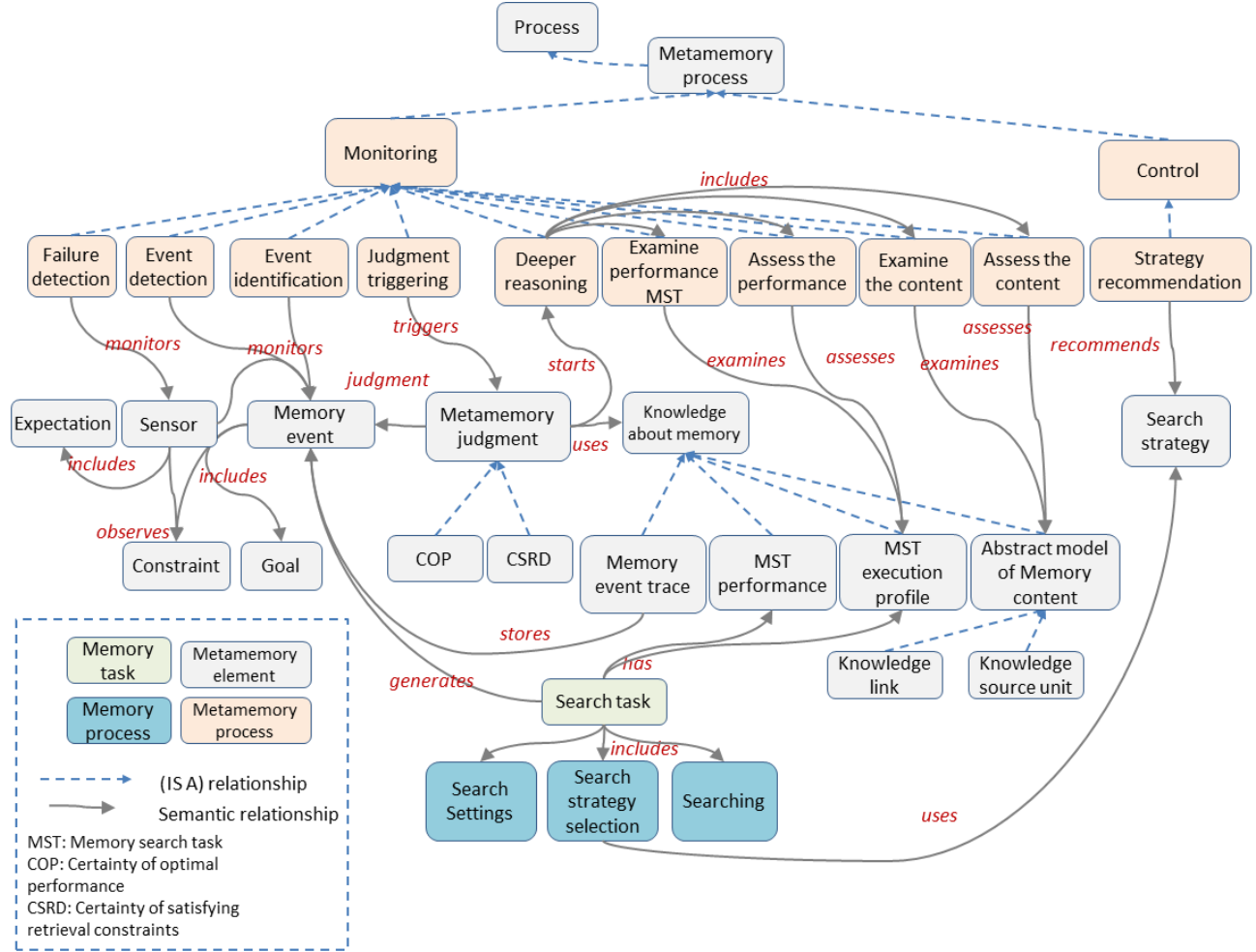


Figure 2. Functional model of M²-Acch

The formal description of M²-Acch functionality has been established using predicate logic in the rest of this section. Due to its generality and richness in specifying models, predicate logic has been used extensively in the literature as a language for describing models. This is done in order to facilitate implementations in various languages like SWRL, Prolog or Python.

2.2.1. Monitoring mechanisms

The proposed model has the following three mechanisms to monitor LTM: *memory event detection*, *metamemory judgment* and *in-depth search*.

2.2.1.1. Memory event detection

In a cognitive system, when a process calls a search task in the memory, then a memory event is triggered. The meta-level stores traces of all the events that occur in LTM.

Rule 1. When some task calls a search task in LTM, then a memory event is triggered.

CT (ct): ct is a task in object-level

$ST(st)$: st is a search task
 $E(e)$: e is a memory event
 $Call(ct, st)$: task ct in object-level calls to search task st
 $T(st, e)$: search task st causes event e to be triggered

$$\forall ct \forall st \exists e (CT(ct) \wedge ST(st) \wedge E(e) \wedge Call(ct, st) \rightarrow T(st, e))$$

Rule 2. All constraints of a search task are associated with the corresponding memory event that is triggered.

$ST(st)$: st is a search task
 $E(e)$: e is a memory event
 $CD(st, d)$: search task st has associated constraint d
 $T(st, e)$: search task st causes event e to be triggered
 $D(e, d)$: constraint d is associated with event e

$$\forall st \forall d \forall e (ST(st) \wedge CD(st, d) \wedge E(e) \wedge T(st, e) \rightarrow D(e, d))$$

Rule 3. If the information constraints of an event are different from the constraints required to execute a search by default, then the meta-level detects a change in the constraints of the event.

$E(e)$: e is a memory event
 $D(e, d)$: constraint d is associated with event e
 $D_f(e, d_f)$: d_f is the expected default constraint (default expectation) for event e
 $CH(e, d)$: change in the constraints d of the event e has been detected

$$\forall e \forall d \exists d_f (E(e) \wedge D(e, d) \wedge D_f(e, d_f) \wedge \neg D_f(e, d) \wedge (d \neq d_f) \rightarrow CH(e, d))$$

2.2.1.2. Reasoning in-depth and Metamemory judgment

Rule 4. If any change in the constraints of an information retrieval task is detected in event memory, then the meta-level decides to launch a deeper reasoning process. The reasoning involves the examination and assessment of the performance of the information retrieval task with similar constraints in the past.

$E(e)$: e is a memory event
 $D(e, d)$: constraint d is associated with event e
 $D(e_h, d_h)$: constraint d_h is associated with event e_h from history trace
 $CH(e, d)$: change in the constraints d of the event e has been detected
 $EH(e_h)$: e_h is an event from event history trace
 $S(s)$: s is a search strategy
 $EHP(e_h, s)$: the event e_h has been processed with strategy s
 $HEHP(e_h, s, p)$: the event e_h has the performance p with strategy s
 $REC(e, s)$: the event e can be processed with strategy s ; with optimal performance

$$\forall e \forall d \exists e_h \exists s (E(e) \wedge D(e, d) \wedge CH(e, d) \wedge EH(e_h) \wedge D(e_h, d) \wedge S(s) \wedge EHP(e_h, s) \wedge HEHP(e_h, s, p) \wedge (p = \text{optimal}) \rightarrow REC(e, s))$$

Rule 5. If at least one strategy that produces an optimal performance is found in the history of events, then a COP judgment with value – high is triggered.

$E(e)$: e is a memory event
 $S(s)$: s is a search strategy
 $REC(e, s)$: the event e can be processed with strategy s ; with optimal performance TJP
 (e, j) : Due to the characteristics of the event e ; COP judgment j is triggered
 $JVP(j, high)$: COP judgment j has value *high*

$$\forall e \forall s \exists j (E(e) \wedge S(s) \wedge REC(e, s) \rightarrow TJP(e, j) \wedge JVP(j, high))$$

Rule 6. If it is not found at least one strategy in the history of events that produces an optimal performance, then it is triggered a COP judgment with value - low.

$E(e)$: e is a memory event
 $S(s)$: s is a search strategy
 $REC(e, s)$: the event e can be processed with strategy s ; having optimal performance
 $TJP(e, j)$: Due to the characteristics of the event e ; COP judgment j is triggered
 $JVP(j, low)$: COP judgment j has value *low*

$$\forall e \forall s \exists j (E(e) \wedge S(s) \wedge \neg REC(e, s) \rightarrow TJP(e, j) \wedge JVP(j, low))$$

Rule 7. If a COP judgment is triggered with a low value, then the meta-level assesses the degree of knowledge that possesses to satisfy the constraint.

$E(e)$: e is a memory event
 $D(e, d)$: constraint d is associated to event e
 $S(s)$: s is a search strategy
 $TJP(e, j)$: Due to the characteristics of the event e ; COP judgment j is triggered
 $JVP(j, low)$: COP judgment j has value *low*
 $RT(e, d, k)$: The system assesses whether it has some kind of knowledge k to satisfy the constraint d for event e

$$\forall e \forall d \forall j \exists k (E(e) \wedge D(e, d) \wedge TJP(e, j) \wedge JVP(j, low) \rightarrow RT(e, d, k))$$

Rule 8. If some kind of knowledge to satisfy the task constraints is retrieved, then the meta-level triggers a CSRJ judgment with value - high.

$E(e)$: e is a memory event
 $D(e, d)$: constraint d is associated to event e
 $TJK(e, j)$: Due to the characteristics of the event e ; CSRJ judgment j is triggered
 $JVK(j, high)$: CSRJ judgment j has value *high*
 $RT(e, d, k)$: The system assesses whether it has some kind of knowledge k to satisfy the constraint d for event e

$$\forall e \forall d \forall k \exists j (E(e) \wedge D(e, d) \wedge RT(e, d, k) \rightarrow TJK(e, j) \wedge JVK(j, high))$$

Rule 9. If no knowledge to satisfy the task constraints is retrieved, then the meta-level triggers a CSRD judgment with value - low.

$E(e)$: e is a memory event

$D(e, d)$: constraint d is associated to event e

$TJK(e, j)$: Due to the characteristics of the event e ; CSRD judgment j is triggered

$JVK(j, high)$: CSRD judgment j has value *high*

$RT(e, d, k)$: The system assesses whether it has some kind of knowledge k to satisfy the constraint d for event e

$$\forall e \forall d \forall k \exists j (E(e) \wedge D(e, d) \wedge \neg RT(e, d, k) \rightarrow TJK(e, j) \wedge JVK(j, low))$$

Rule 10. If observations in sensors do not match with expectations, then there is a violation of expectation. The violations of expectations are interpreted by the meta-level as failures in the reasoning of object-level.

$E(e)$: e is a memory event

$D(e, d)$: constraint d is associated to event e

$SS(s, d)$: sensor s has detected as observation the constraint d

$EXP(s, d)$: sensor s has expectation of constraint d

$V(s)$: violation of expectation in sensor s

$$\forall e \forall d (E(e) \wedge D(e, d) \wedge SS(s, d) \wedge \neg EXP(s, d) \rightarrow V(s))$$

2.2.2. Control mechanisms

Rule 11. If a COP judgment has high value then the strategy with optimal performance in the event history is recommended.

$E(e)$: e is a memory event

$D(e, d)$: constraint d is associated to event e

$S(s)$: s is a search strategy

$TJP(e, j)$: Due to the characteristics of the event e ; COP judgment j is triggered

$JVP(j, high)$: COP judgment j has value *high*

$REC(e, s)$: the event e can be processed with strategy s ; having optimal performance

$R(e, s)$: for event e ; strategy s is recommended the

$$\forall e \forall j \forall d \exists s (E(e) \wedge TJP(e, j) \wedge JVP(j, high) \wedge D(e, d) \wedge S(s) \wedge REC(e, s) \rightarrow R(e, s))$$

Rule 12. If a CSRD judgment has high value then a strategy with support for knowledge in constraints is recommended.

$E(e)$: e is a memory event

$D(e, d)$: constraint d is associated to event e

$TJK(e, j)$: due to the characteristics of the event e ; CSRD judgment j is triggered

JVK (j , $high$): CSRD judgment j has value $high$

RT (e , d , k): The system assesses whether it has some kind of knowledge k to satisfy the constraint d for event e

R (e , s): for event e ; it is recommended the strategy s

SP (s , k): strategy s with support for knowledge k

$$\forall e \forall j \forall d \forall k \exists s (E(e) \wedge S(s) \wedge TJK(e, j) \wedge JVK(j, high) \wedge D(e, d) \wedge RT(e, d, k) \wedge SP(s, k) \rightarrow R(e, s))$$

Rule 13. If the CSRD judgment has low value then it is recommended to stop the memory search task.

ST (st): st is a search task

E (e): e is a memory event

T (st , e): search task st causes event e to be triggered

TJK (e , j): due to the characteristics of the event e ; CSRD judgment j is triggered

JVK (j , low): CSRD judgment j has value low

STP (st): the meta-level recommends stopping search strategy st

$$\forall st \forall e \forall j (ST(st) \wedge T(st, e) \wedge E(e) \wedge TJK(e, j) \wedge JVK(j, low) \rightarrow STP(st))$$

2.3 Information flow in metamemory model (the information layer)

M²-Acch provides support for the monitoring and control processes of information retrieval tasks that are performed in the LTM of an cognitive system. Metamemory functionalities in the meta-level of M²-Acch are activated when some process from object-level calls a search task in LTM, and then triggers a memory event as specified in rule (1). The meta-level stores traces of all the events that occur in LTM.

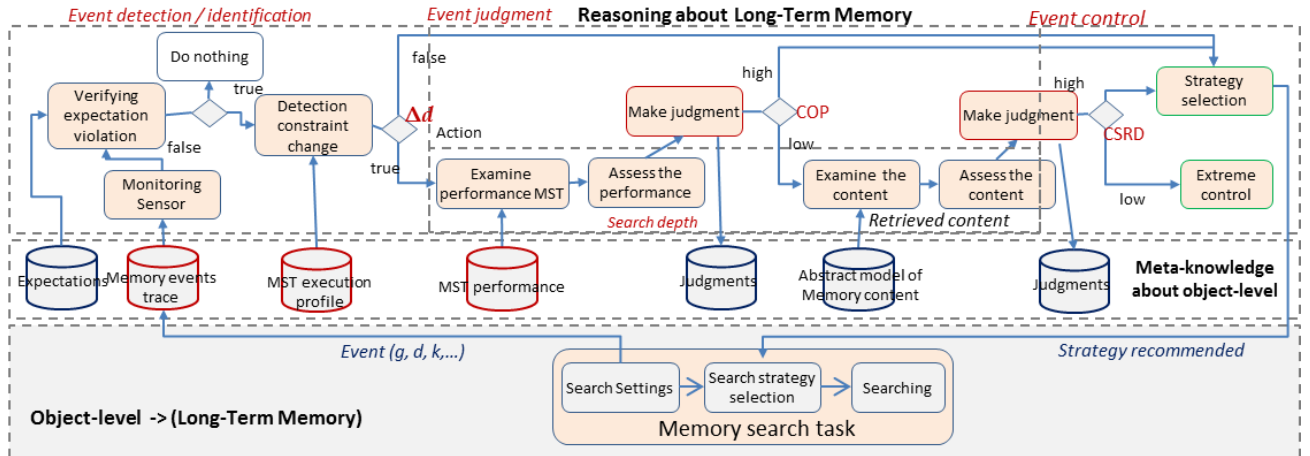


Figure 3. Information flow in M²-Acch

When a new memory event trace is stored in the meta-level, the monitoring process starts as the meta-level detects and identifies the event in LTM.

If the event detected is a call to a search task, then the meta-level checks for changes in task constraints of the target of the search. The changes in the task constraints occur when there is a failure due to difference between the observation and expectation of target of search. Expectations can be specified by default in the system configuration or may be self-generated by the system.

If any change in constraints of the search task is detected, then the meta-level decides to launch a deeper reasoning process about the memory event. The reasoning involves examination and assessment of the performance of the information retrieval task with similar constraints in the past.

In the examination and assessment of the performance process the meta-level searches for events that occurred in the past with similar restrictions.

If the events with similar meta-level constraints are located, meta-level then proceeds to obtain the search strategies that have been used to process such events. If the meta-level has at least one event that has been processed successfully, then it makes a COP judgment with high value. This means that the meta-level has a high level of certainty of knowing the appropriate strategy to satisfy the request of retrieving information contained in the current event. To that purpose the meta-level maintains a performance profile of search tasks, which consists of a record of the search strategies that have been used to process information retrieval requests in the past. In this performance profile, information that shows whether the search strategy could retrieve the required information is also stored.

The meta-level control is based on the value of the metacognitive judgments. For example, if a COP judgment has a high value, then the meta-level recommends the search strategy that has had better performance in events with similar constraints in the past.

In case the judgment has a low value, and the system has available intelligent complex search strategies, then M²-Acch offers the possibility for the meta-level to recommend these strategies. For this purpose, the meta-level evaluates the knowledge about the requirements implicit in the constraints of the search. If some knowledge related to constraint is obtained, then the meta-level triggers a CSRD judgment with high value. Otherwise, the meta-level control mechanism recommends to the object-level to stop the information retrieval, because there is not enough knowledge to process the search.

2.4 Metamemory integration model

Cognitive system

3 Validation and data analysis

3.1 Method

In this section, we describe the features of FUNPRO, the Intelligent Tutoring System designed to assess the M²-Acch metamemory model and how constraint changes are implemented in FUNPRO for information retrieval. Then, the performance metric used for measurement are depicted, followed by a description of overall validation procedure.

3.1.1 Developing of a cognitive system

To validate the metamemory model proposed, we have designed an ITS named FUNPRO (FUNDamentos de PROgramación) using MODESEC¹ methodology (Caro, Toscano, Hernández, & David, 2009). FUNPRO is an Intelligent Tutoring System for teaching *Introduction to Programming in Engineering*, Figure 4.



Figure 4. FUNPRO web environment

3.1.2 Identification of call to retrieve tasks with constraint changes

Figure 5 shows the basic flow which describes the behavior of FUNPRO. Figure 5 has been divided into four sections labeled A, B, C and D; representing different cases of object-level information retrieval tasks.

FUNPRO has implemented three types of search strategies: (1) *matching simple query*, the search query in a simple SQL type; (2) *exclusive search* is similar to (1), but excludes some results and; (3) *vote-based search*, this strategy is based on the nearest neighbor algorithm.

¹ MODESEC is a methodology for design and development of educational software in small teams. MODESEC was developed at University Of Cordoba – Colombia by EdupMedia research group.

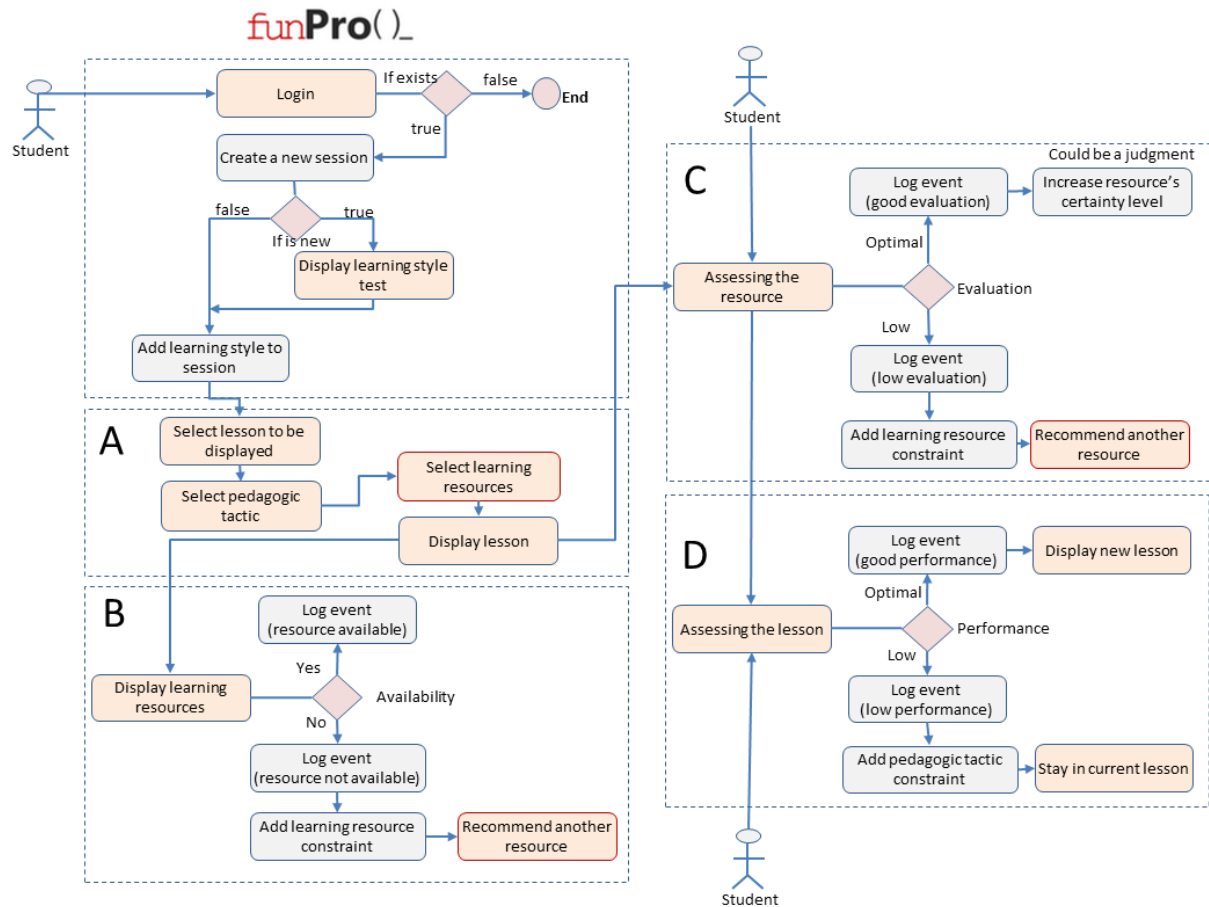
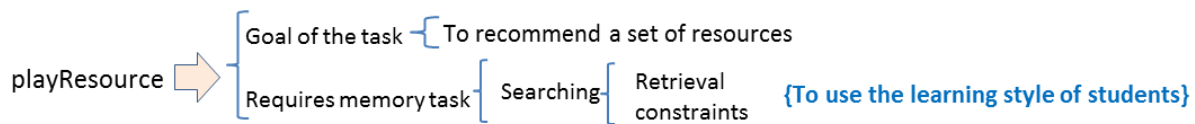


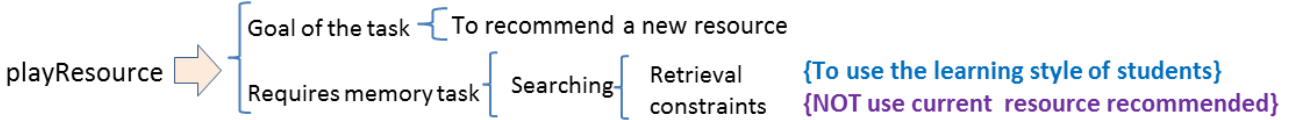
Figure 5. Flow diagram of FUNPRO

FUNPRO has a function called `playResource` that is responsible for retrieving the URL of learning resources from the knowledge base, and deploying them in the lesson. However, the restrictions that FUNPRO generates to search for resources for the lesson are dynamic according to several criteria described below.

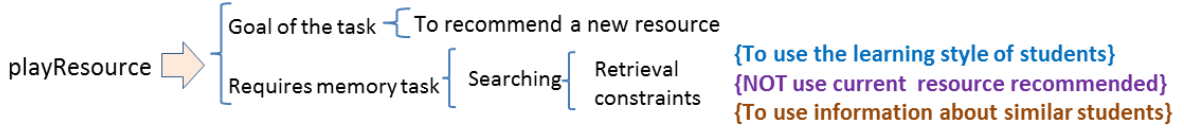
Case A. The student enters the lesson for the first time; therefore FUNPRO has only collected information about the student's learning style to recommends learning resources and teaching strategies for the lesson. Thus `playResource` function receives a single constraint.



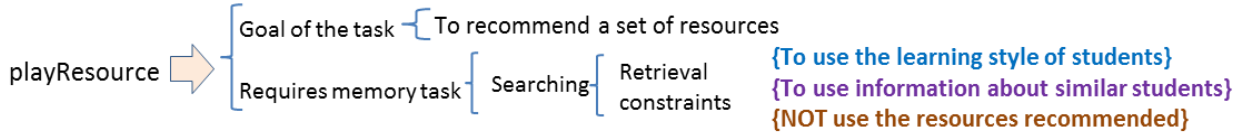
Case B. FUNPRO finds a resource that meets the restrictions of the search, but for some reason cannot be deployed in the lesson, for example: the resource URL is broken. In this case FUNPRO has to retrieve from the knowledge base another resource that supports the student's learning style, but it has to exclude the resource with the URL broken.



Case C. The student evaluates the resource after use. If the resource has received a poor evaluation, the system recommends new resources that have been well evaluated by students with similar characteristics to the current.



Case D. Case D has two variants. In the first, if the student obtains a low performance in the lesson, then the system remains in the current lesson but reconfigures strategies for teaching and learning resources. In the second variant, if the student obtains a high performance; then the system presents a new lesson.



3.1.3 Performance metrics

The validation of the implementation of meta-memory model in FUNPRO on cases A and B are presented next. Since the primary purpose of the M²-Acch is to monitor and control failures in the reasoning process about information retrieval in cognitive systems, we use the reasoning failures dimension as performance metric of the metacognitive capacity of the system. The metric represents retrieval performance (Ghetti, Lyons, Lazzarin, & Cornoldi, 2008) on the number of available resources that were recommended for the lesson. A resource available is one that can be deployed in a lesson, Table I provides a description of the metric for performance evaluation of ITS with metamemory functions.

Table I. Performance metrics

Metric	Description
ART	Total of available resources in retrieval
URT	Total of unavailable resources in retrieval

3.1.4 Process

For validation, we generated 50 student profiles with random assignment of learning styles. Then 400 educational resources profiles were generated, 20 educational resources for each one of the 20 pedagogical tactics supported by FUNPRO. For each student profile, a recommendation of learning resources required for the lesson, based on the learning style was generated.

The simulation of the recommendation process was conducted in eight sessions. In each session the number of unavailable resources in the resource base was gradually increased, see Table II for details. Finally, each session was repeated five times to observe the behavior of the meta-level.

Table II. Session configuration

Session	# of Students	# of resources	# available	# unavailable
1	50	400	360	40
2	50	400	320	80
3	50	400	280	120
4	50	400	240	160
5	50	400	200	200
6	50	400	160	240
7	50	400	120	280
8	50	400	80	320

3.2 Data analysis and discussion

Figure 6 in Section A shows the results obtained in the 8 sessions without the implementation of metamemory in FUNPRO. In this case the average of ART was 71% and the average of URT was 29%. It can be seen that the performance of FUNPRO is decaying with increasing the number of unavailable resources in the resource base.

In Figure 6, section B shows the results obtained in the 8 sessions with the implementation of metamemory in FUNPRO. In this case the average of ART was 98% and the average of URT was 2%. In the worst scenario depicted in session 8 FUNPRO shows an average yield of 94% with respect to the number of recommendations that contain available resources.

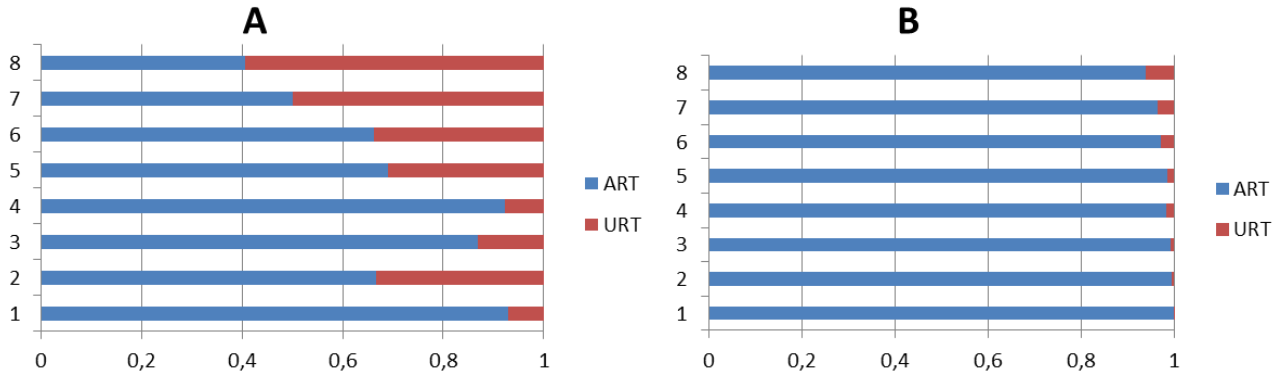


Figure 6. Performance of recommendations in FUNPRO. Section A shows the performance of FUNPRO without metamemory. Section B shows the performance of FUNPRO using M²-Acch.

Figure 7 in Section A shows the results of the comparison between the performance of FUNPRO not using metamemory and using metamemory.

It is noted that when including metamemory, FUNPRO shows a low sensitivity to the progressive increase of unavailable resources in the resource base. This means that FUNPRO can adapt to such situations because it is able to select the appropriate search strategy in the event of failures in the information retrieval.

Thus when performing information retrieval based on the student's learning styles and needed learning resources, then FUNPRO excludes the resource that is not available for future searches.

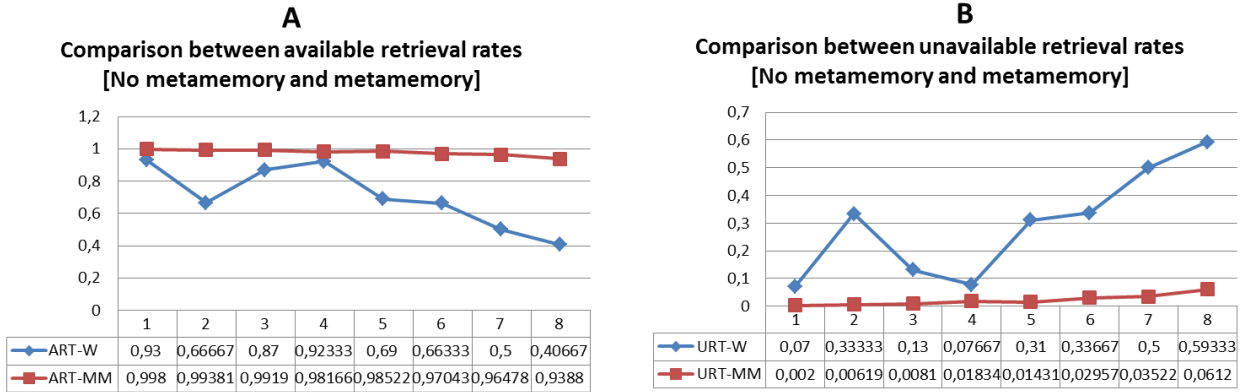


Figure 7. Comparison between retrieval rates in FUNPRO. Section A shows the comparison between available retrieval rates. Section B shows the comparison between unavailable retrieval rates.

The results obtained in the experimental tests show that M²-Acch is able to make adaptations in the search strategies. Adaptations are based on changes in the constraints of information retrieval and allow the system to recognize and prevent failures in the recommendations. Therefore metamemory increases the robustness in terms of failure tolerance in information retrieval from LTM.

4 Conclusion

This paper has presented a novel architecture that involves the use of metamemory for adapting to changes in the constraints of information retrieval tasks from long-term memory.

M²-Acch provides mechanisms for monitoring, identifying and making decisions about the events related to information retrieval that occur in long-term memory. The novel architecture is able to identify changes in constraints affecting information retrieval from long-term memory. M²-Acch has a system based on confidence judgments for recommending search strategies that allow adaptation to changes in the information retrieval constraints.

M²-Acch was implemented in Prolog and validated in an intelligent tutoring system called FUNPRO. The results of the experimental tests show that FUNPRO was able to adapt to changes in the restrictions on the search effort. FUNPRO performance using M²-Acch was superior in terms of retrievals available in comparison to the performance of FUNPRO without M²-Acch. The results of the tests show that M²-Acch is a valid tool for improving the process of information retrieval from long-term memory in cognitive systems.

5 References

- Caro, M. F., Toscano, R. E., Hernández, F., & David, M. E. (2009). Diseño de software educativo basado en competencias. *Ciencia e Ingeniería Neogranadina*, 19(1), 71–98.
- Cox, M., Oates, T., & Perlis, D. (2011). Toward an Integrated Metacognitive Architecture. In P. Langley (Ed.), *Advances in Cognitive Systems: Papers from the 2011 AAI Fall Symposium (FS-11-01)* (pp. 74–81). Technical Report FS-11-01. Menlo Park, CA: AAAI Press.
- Crowder, J. A., & Friess, S. (2010). Metacognition and Metamemory Concepts for AI Systems. In *International Conference on Artificial Intelligence, ICAI 2010*. Las Vegas.
- Davelaar, E. J., Yu, E. C., Harbison, J. I., Hussey, E. K., & Dougherty, M. R. (2013). A rational approach to memory search termination. *Cognitive Systems Research*, 24, 96–103. doi:10.1016/j.cogsys.2012.12.012

- Duch, W., & Szymański, J. (2007). Towards Avatars with Artificial Minds : Role of Semantic Memory. *Journal of Ubiquitous Computing and Intelligence*.
- Düzel, E., Cabeza, R., Picton, T. W., Yonelinas, a P., Scheich, H., Heinze, H. J., & Tulving, E. (1999). Task-related and item-related brain processes of memory retrieval. In *Proceedings of the National Academy of Sciences of the United States of America* (Vol. 96, pp. 1794–9). Retrieved from <http://www.ncbi.nlm.nih.gov/pubmed/20924913>
- Ebbinghaus, H. (1962). *Memory: A contribution to experimental psychology*. (N. Y. D. (Originally published in 1885), Ed.).
- Flavell, J., & Wellman, H. (1977). Metamemory. In & J. W. H. (Eds. . R. V. Kail, Jr. (Ed.), *Perspectives on the Development of Memory and Cognition* (pp. 3-33). Hillsdale, NJ: Lawrence Erlbaum Associates.
- Ghetti, S., Lyons, K. E., Lazzarin, F., & Cornoldi, C. (2008). The development of metamemory monitoring during retrieval: the case of memory strength and memory absence. *Journal of experimental child psychology*, 99(3), 157–81. doi:10.1016/j.jecp.2007.11.001
- Gomes, R. M., Braga, A. P., & Borges, H. E. (2012). Information storage and retrieval analysis of hierarchically coupled associative memories. *Information Sciences*, 195, 175–189. doi:10.1016/j.ins.2012.01.036
- Huet, N., & Mariné, C. (1997). Memory strategies and metamemory knowledge under memory demands change in waiters learners. *European Journal of Psychology of Education*, XII(1), 23–35.
- Kizilirmak, J., Rösler, F., & Khader, P. (2012). Control processes during selective long-term memory retrieval. *NeuroImage*, 59(2), 1830–41. doi:10.1016/j.neuroimage.2011.08.041
- Leake, D. B. (1995). Representing Self-knowledge for Introspection about Memory Search A Planful Framework for Internal Reasoning. In *AAAI Spring Symposium on Representing Mental States and Mechanisms*. Stanford, CA.
- Mecklinger, A. (2010). The control of long-term memory: brain systems and cognitive processes. *Neuroscience and biobehavioral reviews*, 34(7), 1055–65. doi:10.1016/j.neubiorev.2009.11.020
- Metcalf, J., & Dunlosky, J. (2008). Metamemory. In *H. L. Roediger III (ed.), Cognitive Psychology of Memory. Vol. [2] of Learning and Memory: A Comprehensive Reference, 4 vols. (J.*
- Schank, R. (1982). *Dynamic memory: A theory of reminding and learning in computers and people*. (C. U. Press, Ed.). Cambridge, MA.
- Sutcliffe, A., Ennis, M. (1998). Towards a cognitive theory of information retrieval. *Interacting with Computers*, Volume 10, Issue 3, Pages 321-351, ISSN 0953-5438
- Unsworth, N. (2010). On the division of working memory and long-term memory and their relation to intelligence: A latent variable approach. *Acta psychologica*, 134(1), 16–28. doi:10.1016/j.actpsy.2009.11.010

General-Purpose Metacognition Engine

Kenneth M'Balé

KMBALE@CS.UMD.EDU

Darsana Josyula

DARSANA@CS.UMD.EDU

Department of Computer Science, Bowie State University, Bowie, MD 20715 USA

Abstract

The paper presents the architecture of a general-purpose metacognition engine. It is a software agent that collaborates with a host to provide it with metacognitive capabilities. The objective is for the combined system to exhibit adaptive intelligent behavior.

1. Introduction

Artificially intelligent agents are the holy grail of computer science even since visionaries imagined automatons as the ultimate human tool. Intelligence is the ability to bring all the knowledge a system has to bear in the solution of a problem (Newell, 1990). Artificial Intelligence, therefore, is the application of artificial algorithms that use knowledge to solve a problem. An AI system is a biologically inspired attempt to realize or emulate the natural intelligence found in living creatures.

A cognitive agent is software with the ability to create and execute a plan to accomplish goals. An example of a cognitive agent is the software that operates a robot. The agent has the ability to learn and to apply its knowledge to a finite set of problems bound by the design and training it has received. A cognitive agent lacks the ability to reason about its own reasoning. This limitation prevents the agent from breaking through the limits established by its design. For example, a cognitive agent may need to determine when to re-enter learning mode in response to certain types and levels of occurrences of anomalies and failures.

This ability of an agent to monitor its cognitive activities and make adjustments if necessary is termed metacognition. Metacognition is important for agents to adapt to changing environments or deal with surprises or failures. When a perturbation occurs, metacognition can help an agent detect that something is amiss and thus provide an opportunity to deal with the situation. (Anderson & Perlis, 2004) provides a model of metacognition (Metacognitive Loop - MCL) that cycle through (i) detecting anomalies, (ii) analyzing their causes and (iii) providing suggestions to respond to the situation.

At higher levels of abstraction, the different types of expectation violations, types of failures and types of responses that an agent can have are limited. Since the causes of anomalies and failures are finite, it is possible to create a metacognitive agent that provides the cognitive agent with metacognitive abilities. The metacognitive agent analyzes the cognitive agent's reasoning and acts to prevent or correct anomalies and failures normally outside the cognitive agent's adaptability.

This paper presents the architecture of a system that provides general-purpose metacognitive capabilities to any other agent. The proposed *general purpose metacognition engine* (GPME) is a continuation of MCL.

The GPME integrates within a larger system using XML interfaces (M'Balé & Josyula, 2013). The GPME processes streams of observations from the environment. It builds an episodic memory that it uses to project the future from the current observations. Each subsequent moment, it compares actual observations to projected observations. Any discrepancy is an anomaly. The primary goal of the GPME is to minimize the number of anomalies. The GPME responds to each anomaly by suggesting a plan of action to correct it, or by modifying its episodic memory, or both. These responses to a goal related to resolving the anomaly. Our claim is that focusing on anomalies to generate goals while dynamically building a knowledge base of cases results in a system with enhanced adaptability in any environment.

2. Contextual Overview

Figure 1 depicts the global context of the system. The environment contains the host. The host operates within and interacts with the environment. The host contains the GPME. The GPME provides the host with metacognitive capabilities. We refer to the combine host and GPME as the *system*, although this paper is primarily about the GPME.

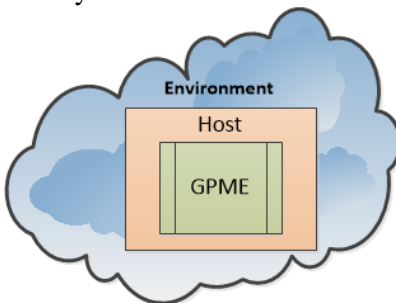


Figure 1 - Global Context

From a logical perspective, Figure 2 depicts that the host is the GPME's interface to the environment. From the GPME's perspective, the host is the environment. More precisely, the GPME only knows of the environment what the host shares. We refer to the flow of information between the GPME and the host as the *telemetry*. Specifically, information flowing from the GPME to the host is a *suggestion* (to act). Information flowing from the host to the GPME is an *observation* (of the environment or of the host itself). We refer to a device and capability the host possesses to interact with the environment as an *instrument*. Specifically, a device or capability to collect information about the environment or about the host is a *sensor*. A device or capability to affect the environment or the host, in a manner detectable by a sensor, is an *actuator*.

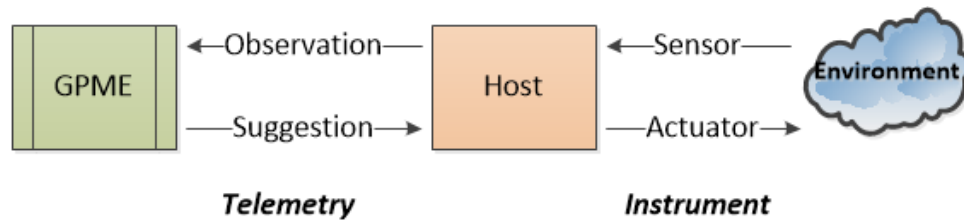


Figure 2 - Global Context Logical View

The host is a sophisticated system capable of performing several functions autonomously. For example, the host is a robot capable of movement, equipped with a gripping arm, and auditory and visual sensors. The host has the ability to safely navigate a space from an original location to a target location. The GPME does not provide detailed step-by-step instruction to navigate from point A to point B. The GPME suggests the host to move from point A to point B. The host is sophisticated enough to act on this suggestion and report its status back to the GPME. We refer to it as a suggestion because the host may not be able to act or may not succeed in the act.

The instruments provide *raw data* from sampling the environment and the host, as well as *processed data* from the sampling. For example, an advanced visual sensor in the host reports an image in PNG format (raw data) and it provides a list of objects or faces it detects in the image (processed data). Therefore, an observation consists of raw data and processed data.

We refer to the individual who is creating the system as the *designer*. The designer incorporates the GPME within her work and designs the host to accomplish her requirements. The designer composes an XML document called the *Environmental Interface Specification* (EIS). The EIS defines the environment to the GPME, including the capabilities of the host. The designer also composes another XML document called the *Operational Interface Specification* (OIS). The OIS specifies the specific communication mechanisms available between the host and the GPME. The EIS and OIS can be revised at any time during the operation of the system.

Therefore, the GPME is a pure intelligence. It is not a human or animal intelligence, and as such, is not designed to interact directly with living things. Since living things are part of the environment, interaction with them is a function of the host. For example, if the system requires human interaction, the host must be equipped with instruments that support human interaction and a human model of world.

2.1 Cognition Circuit

At the core of the GPME, there occurs a continuous cycle called the *Perpetual Cognition Circuit*, depicted in Figure 3. The GPME receives telemetry from the environment, which includes the host. The instruments create an observation. The observation triggers the learning apparatus to process the new observation into the knowledge base. The assimilation of a new observation changes the organization of the episodic memory. The projection apparatus uses the knowledge base to project the future wellbeing of the system. If the future wellbeing of the system is in jeopardy, it suggests actions that maximizes wellbeing and monitors success. The measure of wellbeing is called *homeostasis*. The instinct-level goal of the GPME is to maximize homeostasis.

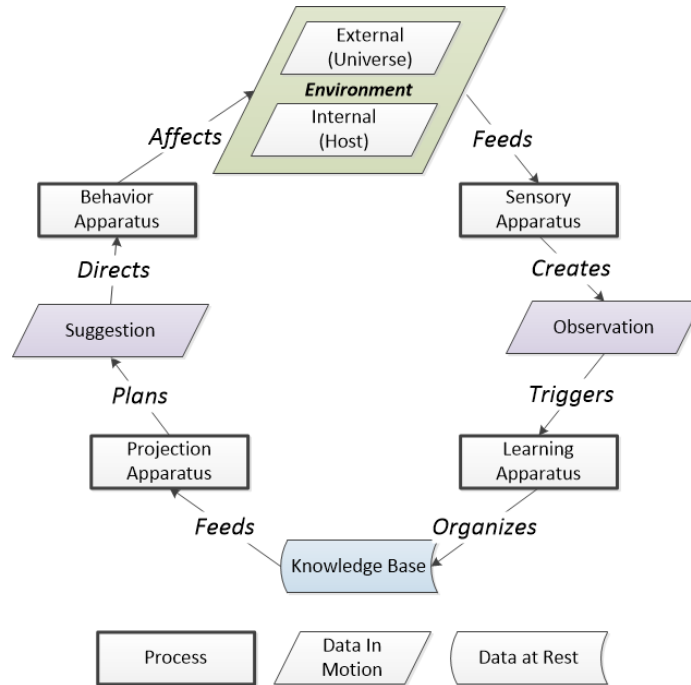


Figure 3 - Perpetual Cognition Circuit

2.2 Learning

In modern software engineering, software is implemented using conditioning or a behaviorist approach. As a result, even intelligent agents are designed and trained to respond to a narrow set of conditions. If circumstances take the agent even slightly out of this conditioning, the agent's performance suffers dramatically (Anderson & Perlis, 2004). Biological agents employ observational learning to mitigate brittleness and to adapt to changing conditions in their environment. As a result, the GPME emulates that approach.

The GPME accepts telemetry about the environment and the host. To filter noise and identify significant features in the telemetry, the GPME looks for rhythmic patterns. This objective requires identifying base and correlated patterns. Section of the telemetry that provide patterns are candidates for learning the relationship between the pattern and the effect on homeostasis.

The GPME has two learning mechanisms with which to build the knowledge base. The direct mechanism processes the telemetry. It is called *progressive reduction*. The indirect mechanism obtains inference rules from other more mature GPME instances. It is called *selective imitation*. The GPME learns from other GPME instances without processing the telemetry that produced the knowledge.

2.3 Internal Base Patterns

From biology, there are at least three candidate internal base patterns.

2.3.1 *Moment*

The first is the perception of time; a circadian rhythm. For example, humans perceive visual intent at less than thirty frames per second. When visual input is presented at a higher frame rate, it does not convey additional information. This observation implies that there is an innate limit to the amount of information we can process at a given time. The limit also defines our smallest perceivable unit of time. We can rationalize smaller divisions of time but we cannot perceive them. Therefore, the GPME needs an internal clock. We refer to an indivisible unit of time, within the GPME, as a *moment*.

2.3.2 *Homeostasis*

The second internal base pattern is the homeostasis. Living creatures have a continuous emotional state that is reflective of perceived success, comfort and various other factors. In the GPME, we refer to these factors as its homeostasis. Homeostasis is a mathematical representation of the GPME's success at functioning and adapting effectively within its environment. An anomaly is a situation where the GPME detects a failure to adapt or a new set of circumstances requiring adaptation. In general, therefore, homeostasis is a function of the number of anomalies the GPME is currently experiencing.

The process of responding to current anomalies or avoiding projected anomalies results in the creation of other goals. These rational-level goals attempt to minimize the number of anomalies and thereby maximize homeostasis. During implementation, several formula variations will be tested to determine the ideal homeostasis formula for a given set of circumstances. In particular, different types of anomalies could be given different weights in the calculation.

Any detected circumstance that jeopardizes homeostasis is called an *anomaly*. To calculate homeostasis at a given moment, the active anomalies are weighted based on their cause or severity. Therefore, the homeostasis value is itself a pattern.

2.3.3 *Emotional State*

The third internal base pattern is the emotional state. The GPME's *emotional state* is a function of the magnitude and cardinality of change in the homeostasis value over time. The emotional state affects several internal processes with the GPME. For example, the emotional state shortens deliberation time or extends deadlines.

Using its clock, homeostasis and emotional state, the GPME can immediately create a rhythm by identifying a pattern in the telemetry and correlating it to any combination of its internal base patterns.

The GPME communicates the emotional state to the host whenever it provides a suggestion. The designer can also specify whether the EIS should communicate a change in emotional state even without a suggestion.

2.4 **Progressive Reduction**

Progressive reduction allows the agent to process high volume streams into manageable clusters where machine learning techniques can be applied to conduct observational learning. In this manner, the GPME is capable of learning and adapting from its environment in a manner very similar to biological agents.

2.4.1 Streams

A unit of telemetry is called a *segment*. An instrument produces one or more segments during a moment. The instrument creates a stream of segments. The combination of all segments captured during a single moment is called a *frame*. Each frame contains several segments from each instrument, including empty segments. Each frame also contains a segment from the GPME stream of suggestions. Each frame also contains a segment from the internal base patterns. The segments share the same moment, the same temporal index. Figure 4 shows the frame vertically with its constituent segments denoted σ .

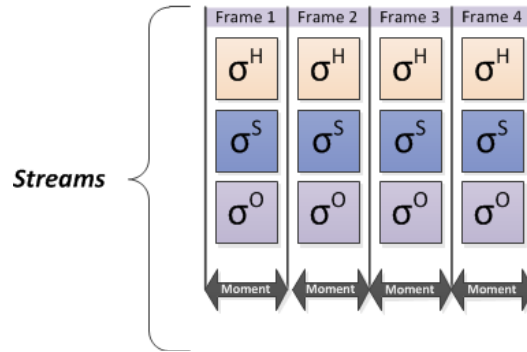


Figure 4 – Streams of Segments and Frames

The telemetry consists of several observation streams (one for each instrument defined in the EIS, collectively denoted σ^o) and the single suggestion stream (denoted σ^s). The frame also contains the GPME's homeostasis and emotional state calculated at that moment (denoted σ^h). The moment indexes the stream of frames. Therefore, at the fundamental level, the GPME processes a stream of frames from the perspective of the host. The stream of frames can also be viewed as a stream of segments from the perspective of an instrument. The GPME is now in a position to analyze the streams and detect patterns within and across them.

We define *short-term memory* as the stream of frames the GPME keeps in its working memory. Short-term memory has a *depth* which is the number of frames or the length of the stream in short-term memory. Refining our earlier definition of emotional state, the calculation uses all frames in short-term memory, including the one from the current moment, to measure the change in homeostasis and establish the current emotional state.

Looking at Figure 4 horizontally, we see the instrument's and internal cycle's stream of segments. The first step in pattern detection is to calculate the probability of occurrence of a segment, based on previous segments in the stream, using the A-distance. A segment with a low probability of occurrence is called a *significant segment*. A significant segment focuses the attention of the learning apparatus. The learning apparatus begins with significant segments and reduces the stream to higher levels of abstractions where it can develop inference rules to project the future state of the streams.

2.4.2 Episodes

Tulving (Tulving, 2005) coined the term episodic memory to refer to the ability to recall specific past events about what happened where and when. Episodic memory is specifically about actual events that occurred in the past. The stream of Figure 4 contains a great deal of information that is needs to be broken into sections for analysis. We refer to a section of temporally contiguous frames from the stream as an *episode*. An episode includes all frames in short-term memory; it is therefore important for short-term memory not to be too large.

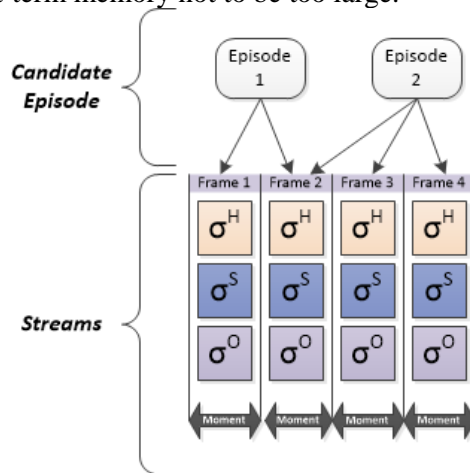


Figure 5 - An Episode consists of a Number of Contiguous Frames

The GPME creates episodes when a type of anomaly occurs; *reflex*, *rational*, *context* and *emotional*. The episode ends when the anomaly no longer exists; either because expectations have caught up with the current state or because the anomalous conditions no longer exist.

The rational method detects that a deadline for achieving a certain homeostasis value has passed, or, that the projected homeostasis value is achieved much sooner than projected. If the homeostasis value is not achieved when expected, the GPME detects the anomaly and creates an episode.

The other types of anomaly detection use the concept of a *bandwidth*. A bandwidth is the projected range of a certain value. The projection is based on historical value contained in the short-term memory. The value is projected to occur within an upper and lower bound. An anomaly occurs whenever the value falls outside the band. The anomaly is *resolved* when the value returns to its original projection. Since the short-term memory changes over time, the bandwidth also changes. Therefore, it is possible for the bandwidth to catch up to the projected value. When this situation occurs, the anomaly is *aborted*.

The reflex method projects an arrival rate of frames for each instrument stream. This projection is called the *instrument arrival rate bandwidth*. For example, the GPME expects the camera to provide an image every five seconds. After six seconds, if an image has not arrived, the GPME detects a reflex anomaly. The same anomaly would also be detected if the image arrived three seconds after the previous one. Since instruments are unlikely to be as regular as the example indicates, the GPME use a range based on its experiences.

The context method detects an anomaly in two different ways. The first way relies on segment significance. The anomaly occurs when a segment that should be significant is not, or, when a

segment that should not be significant is found to be. When the significance of a segment does not match its projected significance, the GPME detects the anomaly and creates an episode. The second way projects the accuracy of the projection. This expectation is called the *projection accuracy bandwidth*. At the most basic level, segment significance is determined by its degree of change from previous states in the segment stream. Therefore, the GPME can detect when a segment that should change has not changed, or a segment that should not change does change, or a when a segment should be empty but contains information. All these conditions make the segment significant.

The emotional method relies on the homeostasis value. The GPME projects the homeostasis value to be within a certain range, called the *homeostasis bandwidth* (See Figure 6). If the homeostasis value is outside the band, the GPME detects the anomaly and creates an episode. The bandwidth is the range between the highest and lowest homeostasis value in short-term memory, however, it is further adjusted by the emotional state.

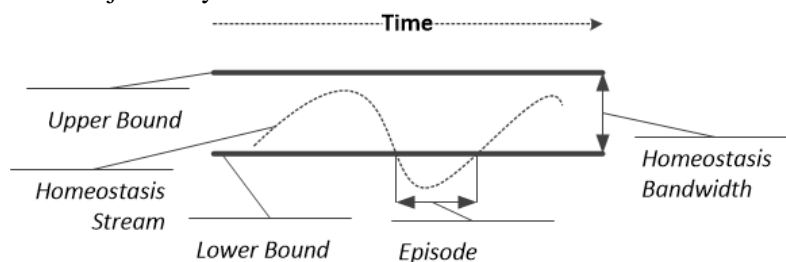


Figure 6 - Homeostasis Bandwidth

For example, the highest and lowest homeostasis in short-term memory is 100 and 150. The bandwidth is 50. The homeostasis of the previous moment is 105. Therefore the unadjusted range is $(105 - 25 =) 85$ to $(105 + 25 =) 130$. Since the GPME is happy, the bandwidth is adjusted by +70% to 85. The adjusted range is $(105 - 43 =) 62$ to $(105 + 43 =) 148$. If the homeostasis value of the current moment falls within the adjusted range, it is normal. A value outside the range is considered anomalous. Note that the anomaly can occur because of exceeding expectations (> 148) or failing to meet expectation (< 62). In other words, an anomaly occurs when the agent is happier than it expected or when it is sadder than it expected. An analogous example could be applied to the instrument arrival rate bandwidth.

The emotional state is a multiplier on the bandwidth. Its value modifies the calculated the bandwidth in the next moment, by shrinking or widening the band. This mechanism emulates biological responses. It enables the GPME to gradually become desensitized and automatically adjust its state of normalcy.

With this information about the detection of anomalies and the creation of episode, we can better articulate the calculation of homeostasis. Homeostasis is a formula that uses the number of active anomalies by type. The formula assigns the highest weight in order; reflex, hardwired (rational), context, emotional and rational (all others besides hardwired). A *hardwired anomaly* is a type of rational anomaly that is the result of a violation of a designer-specified expectation in the EIS.

Therefore, the GPME has the ability to create episodes from the telemetry. As depicted on Figure 5, a newly created episode is referred to as a *candidate episode*. It is a candidate for inclusion in a cluster of episodes. Figure 7 depicts an example of several episodes.

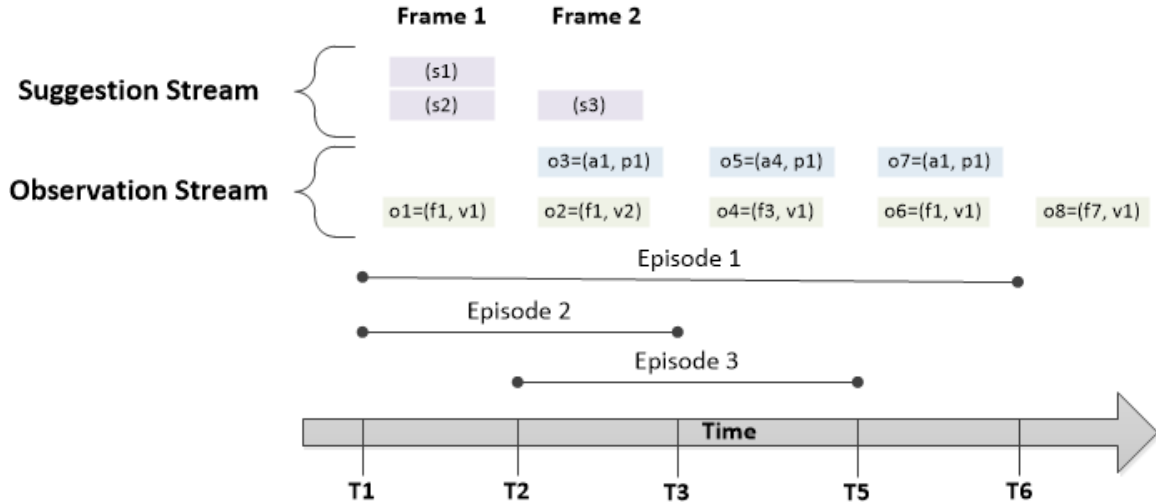


Figure 7 - Example of Episodes

The GPME processes the telemetry (suggestion and observation streams) into frames. In this example, the frames are vertical; s_1 , s_2 and o_1 occurred at the same moment. Two anomalies occurred at moment T_1 resulting in the creation of episode 1 and episode 2. At moment T_3 , another anomaly resulted in the creation of episode 3. At moment T_4 , episode 2 completed. At moment T_7 , episode 3 completed. At moment T_8 , episode 1 completed. At completion, the episode becomes available for learning. Episode 2 consists of frames 1 and 2, which is the same as saying that episode 2 consists of segments (o_1 , o_2 , o_3 , s_1 , s_2 , s_3). Episode 2 and episode 3 overlap in frame 2. Both episodes 2 and 3 are subsets of episode 1.

Let's assume that all episodes were ended by the resolution of the anomaly. Then, we can express the episode as an inference rule: *anomaly + episode* \rightarrow *resolution* where *resolution* = \neg *anomaly*. While the inference rule is obviously true, the episode contains a significant amount of noise. Noise refers to telemetry that had no actual bearing on the anomaly or its resolution. Therefore, the challenge is to find sufficient examples of the same anomaly in order to identify the noise and to determine what part of the telemetry actually affected the anomaly so that we can derive a clean and efficient inference rule.

2.4.3 Cases

Over time, the GPME creates a large number of candidate episodes many of which do not contain any valuable information. To identify the valuable information, the GPME clusters episodes together based on the type of anomaly that created them. Therefore, there are four clusters of episodes; reflex, emotional, hardwired and rational. These four clusters are further divided in terms of their significant segments. Finally, these clusters are further divided in terms of the similarity of the patterns they contain.

In the context of this analysis, rhythmic patterns are of greater significance than the ones that are not. In other words, finding several episodes that have the same rhythmic patterns is significant because this discovery immediately identifies the noise contained within the episodes.

We can talk about the *distance* between two episodes as the quantification of their difference in the terms of the clustering method we described above. Episodes that originate from different anomalies are the most distant from each other. Within the anomaly cluster, the ones with the most different significant segments are most distant. Within the significant segment cluster, the ones with least similar patterns are most distant from each other. Within the similar pattern cluster, the one with the same rhythms are closest to each other. The GPME attempts to create small clusters that contain episodes that have very small distances between them.

Given a sufficiently tight episode cluster, the GPME generates a centroid that is called a *case*. Structurally, a case is identical to an episode. However, it is not an episode because it did not actually originate from the telemetry. It is not an actual experience and cannot be called an episode. Like an episode, the case consists of sequence of frames. The number of frames reflects the number of frames of the cluster episodes. The observation and suggestion segments in the case's frames contain only the significant segments from the cluster episodes. The case derives the homeostasis segments based on the homeostasis values of the cluster episodes. It is now clear that rhythmic patterns are preferred because they result in a higher information gain in identifying which significant segments are actually significant! Since the case only contains significant segments, it is composed of complete, partial and empty frames. We refer to a case's derived frame as a *fragment*. We defined short-term memory earlier. Now, we define *working memory* as the combined short-term memory and projected fragments.

The case is an abstract or pseudo episode created from the significant information in the cluster episodes. The case tells the GPME that, given a certain set of preconditions (from the observation stream), the suggestions (from the suggestion stream) will have a certain effect on homeostasis. As depicted on Figure 8, we refer to this inference rule as the *case predicate*. The GPME uses the case predicate to project the future state of the stream. The future state is expressed in fragments. The telemetry is expressed in frames. A rational anomaly results in a future moment when the frame of that future moment cannot be matched to a fragment projected into that future moment.

The length in moments of the case establishes a deadline for achieving the projected homeostasis value. As we discussed earlier, a rational anomaly results when the number of moments elapses and the projection is not achieved. The availability of predicates and episodic memory make active logics (Elgot-Drapkin, Perlis, Kraus, Miller, & Nirkhe, 1999) highly suitable for managing the agent's responses to expectation violations.

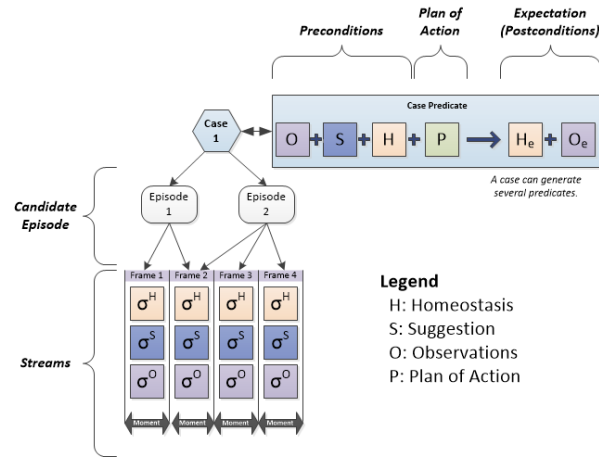


Figure 8 - Cases and Case Predicate

It is important to note that the case can support several predicates, in particular when it is relatively new. The number of predicates arises from the possible combinations of significant segments that arise from their linkages. Further in the document, we will see that frames are connected using several types of links. Thus far, we have only considered the temporal link that is automatically created in the episodic memory. The GPME will need a reasoning mechanism not only to choose a case but also to choose the best predicate for the given situation. We refer to the process of choosing a case to project as the *deliberation* process. To reiterate an earlier point, the current emotional state of the GPME affects the deliberation process.

The GPME sets expectations of its well-being and responds to anomalies by implementing a *plan of action* to return to a normal or better state. The plan of action is derived from the significant suggestions in the episode cluster. The case predicate of Figure 8 creates an *expectation*. The expectation is that, given the preconditions in the telemetry, applying the plan of action results in a projected homeostasis and observation stream. If the expectation is not achieved, the *expectation violation* is a rational anomaly.

Having elaborated on the concept of matching frames to fragments in a given moment, we can measure the accuracy of the projection of a moment as a function of the number of fragments and the number of matched fragments. The projection accuracy bandwidth is determined in the same manner as the homeostasis bandwidth, however, using the highest and lowest moment projection accuracy in short-term memory. The emotional state does not modify the projection accuracy bandwidth.

3. Related Work

The roots of metacognition extend into psychology where research sought to understand its role in development (Yussen, 1985) and learning disabilities. (Wellman, 1990) describes metacognition in humans as a collaboration of several cognitive processes and structures interconnected by the view of self. This theory of mind emerges in childhood as the child

separates itself from its environment and distinguishes between reality and the model of reality in its mind.

In (Cox, 2005), the survey reviews research results of experiments conducted on adult human subjects to determine how they solve problems. The conclusions were that subjects placed in metacognitive conditions perform better because, while problem-focused deliberation leads to good local solutions, metacognition provides the flexibility necessary to discover more complex and efficient solutions by leveraging a more global perspective. When people reflect on their own thought processes as they solve problems, they perform better.

The contribution to this research is the notion that a metacognitive system consists of at least two software components. One, the cognitive component thinks about goals and plans to achieve goals. The other, the metacognitive component thinks about thinking and how it arrives at plans and solutions. The metacognitive component provides for introspection and self-improvement. In most research, this distinction is logical. At the implementation level, the two components are blended into a single agent. However, the GPME is specifically designed to be a separate agent focused on metacognition only.

In (Cox, 2007), a perpetual self-aware cognitive agent is one that fully integrates cognition (planning, understanding and learning) and metacognition (control and monitoring of cognition). In effect, this definition combines the GPME and the host as software components of a single agent. Meta-AQUA is an implementation of this approach using multi-strategy learning. Meta-AQUA uses a form of failure blame assignment called case-based introspection to generate explicit learning goals. A system that detects its knowledge to be flawed must create a specific learning goal to correct the problem.

(Anderson & Oates, 2007) surveys the field and review the emergence of metacognition across several fields and application domains. (Zheng & Horsch, 2005) addresses a key issue; control of computation. Control of computation is to know when to stop a given process. In terms of the metacognitive agent, stopping should occur when the proven best solution to a problem is found. (Nirkhe, 1994) and (Josyula & M'Balé, 2013) further constrain this problem by addressing the problem of time bound reasoning. Reasoning with a deadline refines the meaning of best solution by requiring that the agent can execute the solution while conditions still permit it.

(Anderson & Perlis, 2004) defines brittleness as a system's inability to manage perturbations. A perturbation is any change in the environment or within the system itself that affects its performance in an undesirable way. Perturbation tolerance is the ability to adapt to the conditions by re-establishing the desired performance. Achieving perturbation tolerance requires the system to detect the perturbation and to make targeted adjustments to its configuration. The system must be self-aware and self-guided as it copes with changing conditions. The strategy to achieve perturbation tolerance is called the Metacognitive Loop (MCL). The loop consists of continually noticing the perturbation, assessing the anomaly and guiding a solution into place. MCL enables the system to monitor itself and influence its own performance in real time. It also directs the system to learn when it encounters something it did not know or when it needs to correct beliefs that are now wrong.

Cognitive systems experience three general types of problems; slippage, knowledge representation mismatch and contractions. Slippage refers to ongoing changes to the truth of known facts over time. What is true now is not necessarily true later. Knowledge representation mismatch refers to the problems introduced by representing the same knowledge differently. The representation is not important; the meaning behind the representation is actually what needs to be conveyed. A contradiction occurs when the system simultaneously believes two opposite

beliefs. Metacognition needs to provide general mechanisms to enable the system to overcome these problems.

(Cox, Oates, Paisner, & Perlis, 2012) discusses the application of the A-distance in detecting anomalies in streams of symbolic predicates in the context of the MIDCA cognitive architecture. The method requires the calculation of the probability of occurrence of a predicate at a particular point in the stream. In the paper, a low probability of occurrence is an anomaly that triggers an analysis of the input. In the GPME, the A-distance is used similarly to detect the occurrence of an unexpected segment value in a given instrument stream.

(Elgot-Drapkin et al., 1999) proposes active logics as the formal approach for addressing these problems in a system's knowledge base. Active logics extend first order logic with the concept of Time and Retraction. The result is an episodic logic reasoned that is capable of planning with deadlines, reason with contractions and with changes in the language of discourse.

(Nuxoll & Laird, 2012; Nuxoll, 2007) discusses the application of episodic memory in the context of SOAR. Episodic memory is a history of events that can be used to improve decision-making. Humans have and continually make use of their episodic memories. An episodic memory is:

- Automatic: The system creates an episodic memory automatically.
- Auto noetic: A stored memory is distinct and distinguishable from current sensory activity.
- Temporally indexed: The memory's metadata includes temporal information that orders the memories in order of perception in time.

The conclusion is that episodic memory is essential to sophisticated cognitive capabilities. In particular, the following capabilities are relevant to this research:

- Action Modeling: An agent can use the episodic memory to predict future outcomes.
- Decision Making: The history of success and failure informs future decision making.
- Retroactive Learning: Learning after the fact by replaying or rehearsing events captured in the memories.

Case-based reasoning is closely related to episodic memory. A case describes a problem the system encountered and the solution to the problem (Schank, 1999). The system needs to match a new problem to an existing case to arrive at a previously successful solution. (Geng & Hamilton, 2002) describes a case selection algorithm using rough sets and fuzzy logic. (Chen, Chen, & Su, 2009) describes ontology-based case based reasoning.

(Lim, Suh, & Suh, 2011) discuss the knowledge representation of a robot. Their approach for knowledge representation also combines a graph and an ontology. The key difference between their research and the GPME is that they specifically designed it for a robot (a system) that there is no differentiation between the host and the GPME. The single unified system is designed for human interaction. This fundamental difference results in a substantial difference in the content and use of the knowledge base.

4. Conclusion

In this paper, we have provided an introduction to an agent designed to provide metacognition to any other agent. The GPME accepts any type of telemetry from the host and builds a knowledge base of cases. Each case provides one or more case predicates the GPME uses to establish an

expectation. Expectation violations are a stimulus that causes the GPME to adjust the host's behavior.

Acknowledgements

The authors would like to thank the Bowie State University Department of Computer Science HBGI grant. This work is funded in part by the National Science Foundation grant number HRD-1137541 and HRD-1238784.

References

- Anderson, M. L., & Oates, T. (2007). A review of recent research in metareasoning and metalearning. *AI Magazine*, 17604, 1–17. doi:10.1.1.107.1421
- Anderson, M. L., & Perlis, D. (2004). Logic, Self-awareness and Self-improvement : the Metacognitive Loop and the Problem of Brittleness. *Journal of Logic Computation*, 14(04), 1–20.
- Chen, Y.-J., Chen, Y.-M., & Su, Y.-S. (2009). An Ontology-Based Distributed Case-Based Reasoning for Virtual Enterprises. *2009 International Conference on Complex, Intelligent and Software Intensive Systems*, 128–135. doi:10.1109/CISIS.2009.23
- Cox, M. T. (2005). Metacognition in computation: A selected research review. *Artificial Intelligence*, 169(2), 104–141. doi:10.1016/j.artint.2005.10.009
- Cox, M. T. (2007). Perpetual Self-Aware Cognitive Agents. *AI Magazine*, (2002), 32–51. Retrieved from <http://www.aaai.org/ojs/index.php/aimagazine/article/view/2027/1920>
- Cox, M. T., Oates, T., Paisner, M., & Perlis, D. (2012). Noting Anomalies in Streams of Symbolic Predicates Using A-Distance. *Advances in Cognitive Systems*, 2, 167–184.
- Elgot-Drapkin, J. J., Perlis, D., Kraus, S., Miller, M., & Nirkhe, M. (1999). *Active Logics: A Unified Formal Approach to Episodic Reasoning*.
- Geng, L., & Hamilton, H. J. (2002). ESRS : A Case Selection Algorithm Using Extended Similarity-based Rough Sets. In *2002 IEEE International Conference on Data Mining, 2002. ICDM 2003*. (pp. 609–612). doi:10.1109/ICDM.2002.1184010
- Josyula, D. P., & M'Balé, K. (2013). Timebound Metacognition. In *COGNITIVE 2013*. Valencia, Spain.
- Lim, G. H., Suh, I. H., & Suh, H. (2011). Ontology-Based Unified Robot Knowledge for Service Robots in Indoor Environments. *IEEE Transactions on Systems, Man, and Cybernetics*, 41(3), 492–509.

- M'Balé, K. M., & Josyula, D. (2013). Integrating Metacognition into Artificial Agents. In *AAAI 2013 Fall Symposium Series* (pp. 55–62). Arlington, VA: AAAI Press.
- Newell, A. (1990). *Unified Theories of Cognition*. Cambridge, MA: Harvard University Press.
- Nirkhe, M. (1994). *Time-Situated Reasoning within Tight Deadlines and Realistic Space and Computation Bounds*.
- Nuxoll, A. M. (2007). *Enhancing Intelligent Agents with Episodic Memory*. Retrieved from http://deepblue.lib.umich.edu/bitstream/handle/2027.42/57720/anuxoll_1.pdf;jsessionid=D96DFBFC6BCC1923C0CBEC7C4D4CED7F?sequence=2
- Nuxoll, A. M., & Laird, J. E. (2012). Enhancing intelligent agents with episodic memory. *Cognitive Systems Research, 17-18*, 34–48. doi:10.1016/j.cogsys.2011.10.002
- Schank, R. C. (1999). *Dynamic Memory Revisited* (2nd ed., p. 302). New York, NY: Cambridge Press.
- Tulving, E. (2005). Episodic memory and autoecesis: Uniquely human? In H. S. Terrace & J. Metcalfe (Eds.), *The Missing Link in Cognition* (pp. 4–56). New York, NY: Oxford University Press.
- Wellman, H. F. (1990). *The Child's Theory of Mind*. Cambridge, MA: MIT Press.
- Yussen, S. R. (1985). *The Growth of Reflection in Children*. New York, NY: Academic Press.
- Zheng, J., & Horsch, M. C. (2005). A Decision Theoretic Meta-Reasoner for Constraint Optimization. In *Proceedings of the 18th Canadian Society conference on Advances in Artificial Intelligence* (pp. 53–65). Berlin, Heidelberg: Springer-Verlag. doi:10.1007/11424918_8

Autonomy beyond Anomalies and Goals: A Strategic Perspective

Don Perlis

PERLIS@CS.UMD.EDU

Department of Computer Science, University of Maryland, College Park, MD 20742 USA

Michael T. Cox

MCOX@CS.UMD.EDU

Institute for Advanced Computer Studies, University of Maryland, College Park, MD 20742 USA

Abstract

In recent years there has been strong interest in both reasoning about goal-identification and selection and metacognitive handling of anomalous situations. These two concerns are usually framed in terms of making agents more autonomous and flexible in dynamic and complex domains. Here we wish to argue that there is a natural unifying perspective that includes both concerns and that may point the way to a yet more powerful kind of autonomy.

1. Introduction

An agent often has routine activities in which it is forming and/or following plans in pursuit of existing goals. And there are also situations in which it has to stop and ask itself: what do I do now? One major example of the latter is that of anomaly-handling: something seems out of the ordinary, contrary to expectation, and might indicate the need to do a form of error-correction. This has been the focus of much recent work, for instance Meta-AQUA (Cox & Ram, 1999), the Metacognitive Loop (Anderson & Perlis, 2005), and other similar efforts. Another example is goal-driven autonomy, in which an agent may autonomously alter or add to its goals if circumstances so warrant (Aha, Cox, & Munoz-Avila, 2013; Aha, Klenk, Muñoz-Avila, Ram, & Shapiro, 2010).

We wish to call attention to a level of processing at which an agent considers quite generally what to do: select from among several existing goals, form a plan to achieve an existing goal, continue with a current plan-in-action, alter such a plan, identify a new goal, abandon a plan or a goal, adopt new subgoals in response to unexpected events, explore opportunities for possible goals or other benefits, do a reality-check of beliefs and expectations, and so on. This could perhaps be called the executive level of processing (borrowing that phrase from cognitive psychology), although the terminology already is in use in various cognitive architectures and so might not be the best choice. Instead, let us call this *the strategic level*.

In what follows we briefly sketch some ideas related to the idea of such a processing level.

2. A Sketch of an Approach

We postulate a *metacognitive monitoring activity (MMAC)* that runs in parallel with an agent's normal routine activity of planning-acting in pursuit of already-identified goals. MMAC will be

aware of such routine activities that are underway, and also of their aims and expectations, and of how (at least some) events are actually unfolding (which may or may not be as expected). As MMAC processes this real-time information, it also asks itself over and over: What should I do now? What choices are there? Is there anything that would be better to do than what I am doing? MMAC would normally run in the background, unless something pops into prominence in virtue of a certain salience or threshold that is reached. What can govern such an event?

One way to envision this is in terms of the A-distance (Kifer, Ben-David, & Gehrke, 2004), which assesses alterations in time-series data that exceed a given threshold. This is a crucial kind of hedging-factor. For any given set of expectations will almost certainly fail to be fully identical to observed events. Tiny variations are the norm, and one cannot possibly attend to all of them (nor would it make sense to do so if it were possible). Yet how can such thresholds be determined, when context means everything? In some contexts, a small variation in color or noise-level may be insignificant, and in others may flag major problems or opportunities.

We think that learning is a promising approach here: an agent can learn, for a given context in which it may be operating (or planning to operate in), which are the important things to attend to. This can be partly at the explicit symbolic level (e.g., a teacher can tell the agent some items to watch for and some to ignore) and partly subsymbolic (experience can provide ranges of “normalcy” that the agent trains into its routines.¹

Now, we doubt A-distance alone will be enough to cover all the cases that we envision for MMAC. For instance, another agent might simply tell our agent that something is important. That is unlikely to cross an A-distance threshold, since conversations may go on all the time, with words flowing rapidly back and forth. It would presumably require a rather high-level reasoning process to understand language sufficiently well to distinguish in a general principled way between, say, “such matters are unimportant” and “don’t ever assume that such matters are unimportant.” Thus we hypothesize that the strategic level of MMAC is a complex organizer of cognition that integrates activities and seeks to improve their lot.

3. Conclusion

In this paper, we suggest that an underlying research issue exists of considerable potential for enhanced autonomy: *how to design an agent with an effective and general-purpose “what do I do now” capacity*. The capacity bears on many cognitive processes and seems crucial for high-level reasoning in complex ever-changing environments. Researchers have at times studied aspects of what we describe under the MMAC banner in terms of metacognition. Other researchers have examined some of these issues in terms of goal reasoning and goal-driven autonomy. It may be the case that we are all speaking of the same process.

Acknowledgements

This material is based upon work supported by ONR Grants # N00014-12-1-0430 and # N00014-12-1-0172 and by ARO Grant # W911NF-12-1-0471.

¹ A-distance was developed largely for the latter situation, with continuous real-valued data; but recent work has shown that it also is effective for discrete symbolic data (Cox, Oates, Paisner, & Perlis, 2013).

References

- Aha, D. W., Cox, M. T., & Muñoz-Avila, H. (Eds.) (2013). *Goal reasoning: Papers from the ACS workshop* (Tech. Rep. No. CS-TR-5029). College Park, MD: University of Maryland, Department of Computer Science.
- Aha, D. W., Klenk, M., Muñoz-Avila, H., Ram, A., & Shapiro, D. (Eds.) (2010). *Goal-Directed Autonomy: Papers from the AAI Workshop*. Menlo Park, CA: AAAI Press.
- Anderson, M., & Perlis, D. (2005). Logic, self-awareness and self-improvement. *Journal of Logic and Computation* 15, 21–40.
- Cox, M. T., Oates, T., Paisner, M., & Perlis, D. (2013). Detecting change in diverse symbolic worlds. In L. Correia, L. P. Reis, L. M. Gomes, H. Guerra, & P. Cardoso (Eds.), *Advances in Artificial Intelligence, 16th Portuguese Conference on Artificial Intelligence* (pp. 179-190). University of the Azores, Portugal: CMATI.
- Cox, M. T., & Ram, A. (1999). Introspective multistrategy learning: On the construction of learning strategies. *Artificial Intelligence*, 112, 1-55.
- Kifer, D., Ben-David, S., & Gehrke, J. (2004). Detecting change in data streams. *Proceedings of the Thirtieth Very Large Databases Conference* (pp. 180-191).

Towards A Unified Framework for Learning and Processing Perceptual, Relational, and Meta Knowledge

Marc Pickett

MARCPICKETT1@GMAIL.COM

NRC Postdoctoral Fellow, Naval Research Laboratory, Washington, DC

Abstract

We present a framework for representing, learning, and processing meta-knowledge. Our framework leverages a system initially developed for perceptual learning and processing to process both relational structures in general and feature hierarchies in particular. We describe preliminary results demonstrating how our system can be used to learn *meta-ontologies*, or feature hierarchies of feature hierarchies.

1. Introduction

A major difference between the minds of humans and other mammals is humans' ability to perform metacognition. Though there is debate about the level of non-human mammals' metacognitive capabilities, it is generally agreed that humans' abilities vastly surpass those of other animals in this area (Carruthers, 2008; Smith, 2009). However, in terms of gross neuroanatomy, human brains seem to have no special structures or mechanisms that are absent in the brains of simpler mammals, such as rabbits, that have little cognitive capacity beyond perception and action (Roth & Dicke, 2005). The chief difference between human brains and those of other mammals is that humans have a vastly expanded neocortex (Rilling, 2006). Furthermore, there is evidence that the neocortex in newborns is both uniform and plastic, with differentiation arising through learning from experiences (Sur & Rubenstein, 2005; Mountcastle, 1978). That is, the a newborn's neocortex is the same basic mechanism repeated many times, and it is this mechanism, which we call the *cortical substrate*, that accounts for the bulk of human learning and reasoning.

From this evidence, we adopt the hypothesis that it is possible to build an intelligent "newborn" agent using only a handful of basic mechanisms. If this hypothesis is true, such an elegant design is attractive for researchers studying cognitive systems because creating an agent with human-level intelligence would entail an implementation of only a handful of mechanisms (and allowing the system to develop its representations through learning) rather than specialized mechanisms for each of the myriad aspects of human intelligence.

Although our work is not constrained by biological plausibility, algorithms loosely based on the neocortex have emerged that have desirable properties for intelligent agents. For example, cortically-inspired models have achieved state-of-the-art performance for computer vision (Le et al., 2012) and some classification tasks (Chandrashekar & Granger, 2011) by learning feature hierarchies.

If an expanded neocortex accounts for the bulk of the cognitive differences between humans and other mammals, then an open question is how an expanded neocortex might account for these differences. That is, an account is missing of how a cortical substrate can be leveraged to account for higher level cognition, such as symbolic reasoning, analogical inference, and metacognition (Granger, 2011).

In previous work, we showed how a cortically-inspired algorithm could account for analogical inference (Pickett & Aha, 2013b). In this paper, we present preliminary results in our attempt to extend our earlier framework to support meta-knowledge, with the belief that this will be useful for metacognitive processing. In particular, we focus on the the question of how knowledge *about* feature hierarchies can be encoded as a feature hierarchy.

First, we provide background on *Ontol*, a model loosely based on the cortical substrate, and how it can be used to process both perceptual and relational data (Section 2). We then show how *Ontol* can be used to build a hierarchy of feature hierarchies (Section 3). Finally, in Section 4 we discuss shortcomings of our framework and how it might be applied to metacognition, and then conclude (Section 5).

2. Background on Learning Feature Hierarchies

Our current model for the cortical substrate, *Ontol* (Pickett, 2011), is a pair of algorithms, both of which are given “sensor” inputs (fixed-length, real-valued non-negative vectors). The first algorithm, *chunk*, constructs a feature hierarchy, or *ontology*, that concisely encodes the inputs. For example, given a set of vectors representing visual windows from natural images, *Ontol* produces a feature hierarchy loosely modeled on that seen in the visual cortex. The second algorithm, *parse*, takes as input an ontology (produced by the first algorithm) and a new vector, and *parses* the vector. That is, it produces as output the new vector encoded in the higher-level features of the ontology. In addition to “bottom-up” parsing, the second algorithm also makes “top-down” predictions about any unspecified values in the vector.

Ontol is ignorant of the modality of its input. That is, *Ontol* is given no information about what sensory organ is producing its inputs. Because of this ignorance, we are able to leverage *Ontol* to find patterns in abstract “sensory” inputs that are actually encodings of relational structures.

Figure 1a shows the ontology constructed by *chunk* when applied to an animal dataset (from Blake & Merz, 1998), where the “sensory percepts” are features for each animal. For simplicity, in this example and the remainder of the paper, we will consider a simplified version of *Ontol* that takes as inputs *feature bags*, which are mathematically equivalent to sparse vectors with positive integer values.

In earlier work we demonstrated how relational structures, such as the “Sour Grapes” story (from Thagard et al., 1990) shown in Figure 2, can be represented as feature bags such that they could be given to *Ontol* as input. When given a collection of stories represented this way, *Ontol* learned a hierarchy of plot devices (shown in Figure 3) that could be used to efficiently retrieve analogs for new stories (Pickett & Aha, 2013a) and perform analogical inference (Pickett & Aha, 2013b).

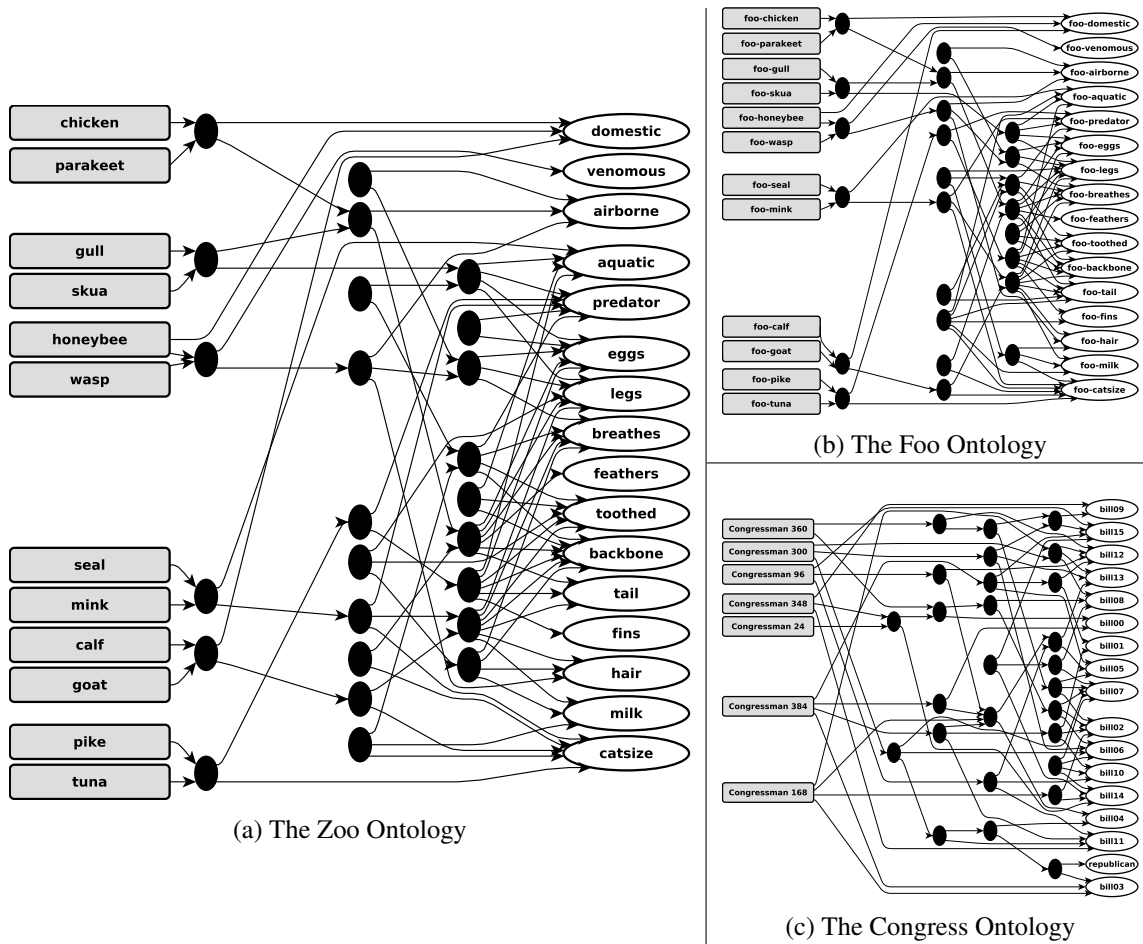


Figure 1: Structurally Similar Ontologies. Instances are shown on the left, and base-level features are on the right. Black nodes in the middle correspond to higher-level features. For example, in Figure 1a, the `chicken` is described by the black node roughly corresponding to “domestic fowl”, which, in turn, is described as both `domestic` and by the node that roughly corresponds to “bird”.

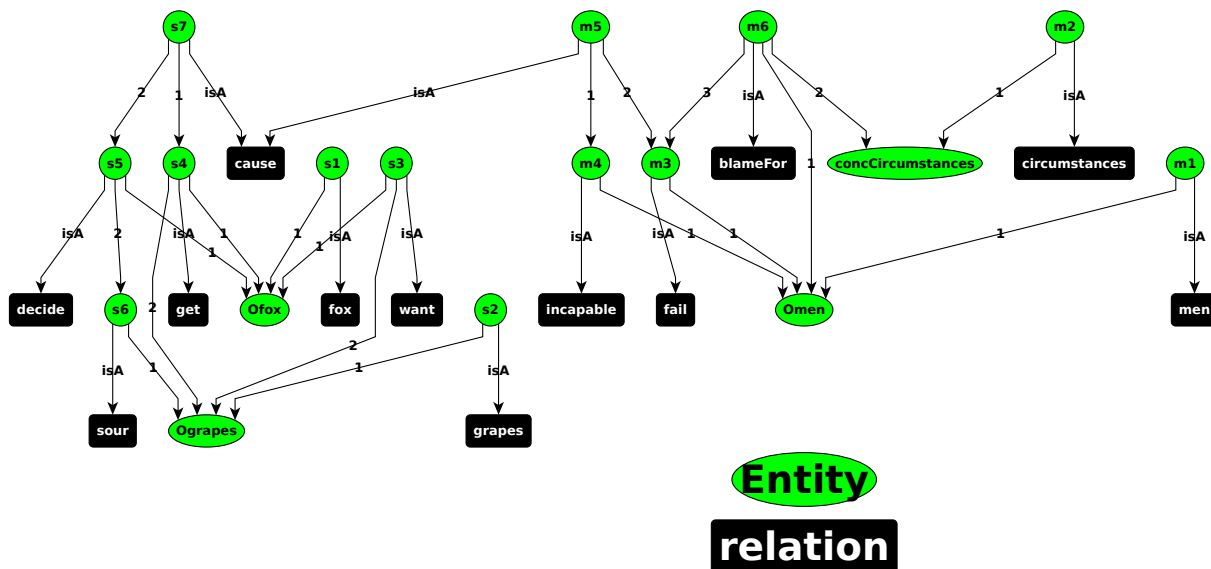


Figure 2: The “Sour Grapes” Relational Structure. This fable was converted into predicate form by Thagard et al. 1990, and displayed here as a graph, where relations are shown as black rectangles with numbered edges and entities are shown as green ellipses. For example, the structure beginning with the node at top center-left marked as `m6` represents the statement `(blameFor Omen concCircumstances m3)`, which can further be expanded to `(blameFor Omen concCircumstances (fail Omen))`.

3. Learning Hierarchies of Feature Hierarchies

The key insight for this paper is that ontologies, such as those shown in Figures 1a and 3, themselves are relational structures. Therefore, we can encode these structures using a method similar to that for encoding stories, and use *Ontol* to learn an ontology that describes a group of ontologies. Although many methods have been proposed for representing meta-knowledge (for a survey see Cox, 2005), to our knowledge, the work presented here is the first representation that uses a feature hierarchy to represent both perceptual knowledge, higher-order knowledge, and knowledge about knowledge.

We now describe a method for transforming ontologies into feature bags so that these can be given as input to *Ontol*. The first intuition behind this transform is that a feature hierarchy essentially encodes relations among the nodes in it. The second intuition is that an encoding of relations among n entities can be approximated by describing the relations among the entities in each of many overlapping subsets of n . For example, if *hair* is highly correlated with event *milk* and *milk* is highly correlated with *warmblooded*, then it is often the case that *hair* and *milk* together predict *warmblooded*. For the current implementation, we use tallies over truth values to represent relations. For example, the 3-way relation among *hair*, *milk*, and *warmblooded* would be represented by a truth table with a tally for each of the 2^3 truth assignment possibilities.

The size of the truth tally is exponential in the number of variables. Therefore, we break each large relational structure into multiple overlapping *subsets* of nodes. Our algorithm exploits a prin-

ciple akin to one used by the HMax model of the visual cortex (Riesenhuber & Poggio, 1999): as the number of subsets for a relational structure increases, the probability decreases that another structure has the same subsets without being isomorphic to the first.

The process for learning a meta-ontology from a set of ontologies, called `metaOntol`, is described in Figure 1. This algorithm extracts t subsets from each ontology, transforms them into canonicalized truth tables. Treating each truth table as a feature bag and `metaOntol` chunks these feature bags to create an ontology of truth tallies called *truthTallyOntology*. `metaOntol` then re-encodes the truth tallies by parsing them using this ontology, and re-encodes the original ontologies (from which the subsets came) as a feature bag of the parsed windows. Finally, `metaOntol` runs another pass of `Ontol`'s chunking algorithm on the re-encoded structures to generate the meta-ontology. Note that since `Ontol` is ignorant of the modality of its input, it processes this meta-knowledge exactly the same way it processes any other type of knowledge.

As a preliminary proof of concept, we created a simple ontology of ontologies by first learning 6 object-level ontologies, transforming these to feature bags, then learning a meta-ontology from the transformed ontologies. The 6 object-level ontologies included the Zoo ontology shown in Figure 1a, a ‘‘Congress’’ ontology partially shown in Figure 1c, which `Ontol` learned from a dataset of 435 congressmen’s votes on 16 bills in 1984. We also learned two additional ontologies from copies of each of these datasets, with the exception that the instances and features have been renamed. For example, from the ‘‘zoo’’ dataset we created the ‘‘foo’’ dataset in which a `foo-chicken` is `foo-domestic`. As shown in Figure 1b, the resulting ontologies are structurally similar (though not completely isomorphic due to randomization effects) to the original ontologies, but superficially dissimilar because they share no common nodes. (We assume that for our system, the atomic symbols `foo-domestic` and `domestic` look no more alike than any other pair of symbols.)

Given these 6 ontologies (and somewhat arbitrarily choosing the size of truth tally $m = 4$, and the number of truth tallies per ontology $t = 100$), `metaOntol` transformed each into a bag of features using the method described above. Using this representation `Ontol` learned the ontology shown in Figure 4. This meta-ontology successfully groups together the structurally similar but superficially different ‘‘zoo’’, ‘‘foo’’, and ‘‘voo’’ ontologies, as well as the variations of the ‘‘congress’’ ontologies.

4. Discussion

The framework we have presented, using cortically-inspired models to represent knowledge about cortical models, is still new and currently has much room for growth. In particular, further work is needed to investigate *how* meta-ontologies might be used by an intelligent agent to accomplish its goals. We also discuss how the current framework might be improved.

It is interesting to consider how the meta-ontology in Figure 4 might be used. Since the meta-ontology captures structural, rather than surface overlap, the meta-ontology might be used for analogical knowledge transfer between object-level ontologies. For example, a similar meta-ontology might be used to find invariances in computer vision and other modalities by finding areas that are superficially different yet behaviorally similar. (E.g., by noticing that the feature hierarchy describing the top left of the visual field is roughly isomorphic to the hierarchy describing the bottom right, a system could discover translation invariance.)

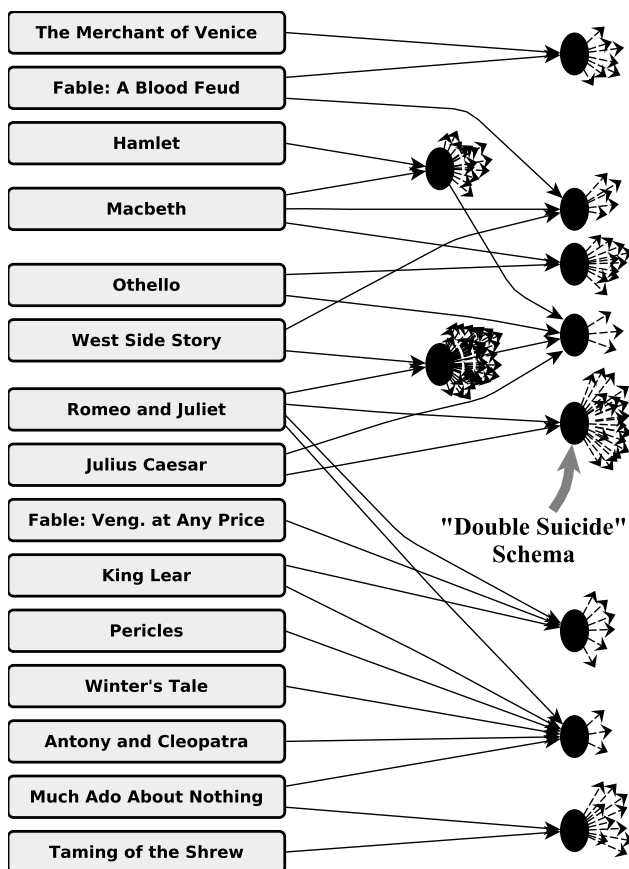


Figure 3: Part of the ontology learned from a story dataset. As in the Zoo Ontology in Figure 1a, black ovals represent higher level concepts. While in the Zoo Ontology, higher level concepts correspond to shared surface features, in this figure, high level concepts correspond to shared structural features (plot devices). For example, the denoted oval on the right represents a *Double Suicide* schema, which happens in both *Romeo and Juliet* and in *Julius Caesar*.

Table 1: Algorithm for Learning a Meta-ontology

```

// Input is  $\Omega$ , a set of ontologies and  $I$ , a set of instances
//  $m$  is the size of truth tally, and  $t$  is the number of truth tallies per ontology
define metaOntol( $\Omega, I, m, t$ ):
  // Create an ontology  $C$  of canonicalized truth tallies.
  let allTallies be an empty list.
  // Get the truth tallies.
  foreach  $\omega \in \Omega$ :
    let tallies $_{\omega}$  be an empty list.
    // Get  $t$  truth tallies.
    repeat  $t$  times:
      let truthCounts be a hash tally initialized to all zeroes.
      let smallont = randomly chosen subset of  $\omega$  of size  $m$ 
      for  $i \in I$ :
        let on $_i$  = the set of nodes in smallont activated when using  $\omega$  to parse  $i$ 
        truthCounts[on $_i$ ] ++ // Make tally for truth entries
        add canonicalize(truthCounts, smallont) to allTallies and to tallies $_{\omega}$ 
  // Chunk the truth tallies
  let truthTallyOntology = chunk(allTallies)

  // Represent each  $\omega$  as a count of parsed tallies
  let  $\Omega$ asSet be an initially empty list.
  foreach  $\omega \in \Omega$ :
    let wasTallyOfTallies be an initially empty set.
    foreach truthCounts  $\in$  tallies $_{\omega}$ :
      parse truthCounts using truthTallyOntology and add the parse to wasTallyOfTallies
    add wasTallyOfTallies to  $\Omega$ asSet

  // Chunk the set of ontologies represented as feature bags.
  return chunk( $\Omega$ asSet)

// Routine to canonicalize a truth tally.
// This tries all  $m!$  orderings of smallont
define canonicalize(truthCounts, smallont):
  // Find the ordering that yields the truth-tally that comes first ‘‘alphabetically’’.
  let bestCounts = truthCounts
  foreach ordering in all  $m!$  orderings of smallont:
    make a new truth tally truthCountsNew substituting ordering for smallont
    if comesFirst(truthCountsNew, bestCounts)
  return bestCounts

// Comparing two isomorphic truth tallies.
define comesFirst( $a, b$ ):
  let keys be the sorted keys from the union of  $a$ ’s and  $b$ ’s keys
  foreach key  $\in$  keys:
    if  $a[\textit{key}] \neq b[\textit{key}]$ : return  $a[\textit{key}] \neq b[\textit{key}]$ 
  return false

```

Motor controllers, such as traces of arm movements, can be encoded in the cortex of the cerebellum (Albus, 1971). Conceivably, we can create an ontology of such motor controllers and leverage

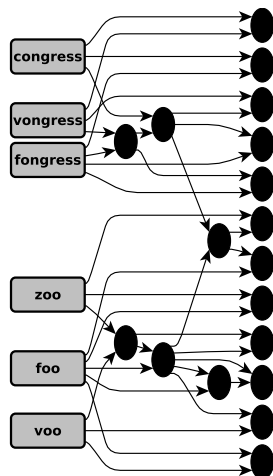


Figure 4: An ontology of ontologies

this ontology to quickly learn new motor skills. If traces of thought can be described as predicate logic, then we can leverage our framework to build an “ontology of thought processes” by transforming predicate logic into a relational structure and then into feature bags. Ontol can then find patterns in these feature bags.

This latter approach may be a promising direction for research in metacognition, as it offers a potential way for a system to analyze its own cognitive processes. This would entail first developing a way to represent cognitive processes themselves as cortical substrate (perhaps in a manner analogous to how motor processes can be represented as sections of cerebellar cortex), then using the current framework to find patterns in the cognitive processes, and finally using these patterns to improve the cognitive processes.

There is room for improvement in both our cortical model and the method for transforming relational structures in general, and ontologies in particular, into feature bags. In its current incarnation, Ontol builds a hierarchy of conjunctions. Significant leverage can be had by including nodes that represent disjunctions (Riesenhuber & Poggio, 1999) and sequences (George & Hawkins, 2009). Including disjunctions will not significantly affect our current transformation algorithm, since truth tallies can be computed using essentially the same method. However, the addition of sequences will require different ways of expressing relations among nodes, such as graphlet kernels (Shervashidze et al., 2009). *metaOntol* currently assumes that the separate ontologies are both segmented and within a manageable size. In reality, ontologies such as semantic networks tend to form contiguous networks (Steyvers & Tenenbaum, 2005). To address both problems, a future version of our transformation will grab a hierarchy of overlapping bounded-size contiguous subgraphs from the ontology in a manner analogous to overlapping receptive fields in computer vision.

5. Conclusion

In this paper we have shown preliminary results demonstrating how Ontol, a system capable of learning feature hierarchies, can learn a meta-ontology, or hierarchy of feature hierarchies. This work is still in its early stages, and future work includes detailing *how* meta-ontologies can be used for knowledge transfer, analogical reasoning, invariance discovery, and meta-cognition.

References

- Albus, J. S. (1971). A theory of cerebellar function. *Mathematical Biosciences*, 10, 25–61.
- Blake, C., & Merz, C. (1998). UCI Repository of Machine Learning Databases.
- Carruthers, P. (2008). Meta-cognition in animals: A skeptical look. *Mind & Language*, 23, 58–89.
- Chandrashekar, A., & Granger, R. (2011). Derivation of a novel efficient supervised learning algorithm from cortical-subcortical loops. *Frontiers in computational neuroscience*, 5.
- Cox, M. T. (2005). Metacognition in computation: A selected research review. *Artificial Intelligence*, 169, 104–141.
- George, D., & Hawkins, J. (2009). Towards a Mathematical Theory of Cortical Micro-circuits. *PLoS Comput Biol*, 5.
- Granger, R. (2011). How brains are built: Principles of computational neuroscience. *Cerebrum; The Dana Foundation*.
- Le, Q., Ranzato, M., Monga, R., Devin, M., Chen, K., Corrado, G., Dean, J., & Ng, A. (2012). Building High-Level Features using Large Scale Unsupervised Learning. *International Conference in Machine Learning*.
- Mountcastle, V. (1978). An Organizing Principle for Cerebral Function: The Unit Model and the Distributed System. *The Mindful Brain*, 7–50.
- Pickett, M. (2011). *Towards Relational Concept Formation From Undifferentiated Sensor Data*. Doctoral dissertation, University of Maryland Baltimore County.
- Pickett, M., & Aha, D. (2013a). Spontaneous Analogy by Piggybacking on a Perceptual System. *Proceedings of the 35th Annual Conference of the Cognitive Science Society* (pp. 3229–3234). Austin, TX.
- Pickett, M., & Aha, D. W. (2013b). Using cortically-inspired algorithms for analogical learning and reasoning. *Biologically Inspired Cognitive Architectures*, 6, 76–86.
- Riesenhuber, M., & Poggio, T. (1999). Hierarchical Models of Object Recognition in Cortex. *Nature Neuroscience*, 2, 1019–1025.
- Rilling, J. K. (2006). Human and Nonhuman Primate Brains: Are they Allometrically Scaled Versions of the Same Design? *Evolutionary Anthropology: Issues, News, and Reviews*, 15, 65–77.
- Roth, G., & Dicke, U. (2005). Evolution of the Brain and Intelligence. *Trends in Cognitive Sciences*, 9, 250–257.
- Shervashidze, N., Vishwanathan, S., Petri, T., Mehlhorn, K., & Borgwardt, K. (2009). Efficient Graphlet Kernels for Large Graph Comparison. *Int. Conf. on AI & Stats.*
- Smith, J. D. (2009). The study of animal metacognition. *Trends in cognitive sciences*, 13, 389–396.

- Steyvers, M., & Tenenbaum, J. B. (2005). The large-scale structure of semantic networks: Statistical analyses and a model of semantic growth. *Cognitive science*, *29*, 41–78.
- Sur, M., & Rubenstein, J. L. (2005). Patterning and plasticity of the cerebral cortex. *Science Signaling*, *310*, 805.
- Thagard, P., Holyoak, K., Nelson, G., & Gochfeld, D. (1990). Analog Retrieval by Constraint Satisfaction. *Artificial Intelligence*, *46*, 259–310.

An Architecture for Flexible Problem Solving

Pat Langley

PATRICK.W.LANGLEY@GMAIL.COM

Miranda Emery

MEME011@AUCKLANDUNI.AC.NZ

Michael Barley

MBAR098@CS.AUCKLAND.AC.NZ

Department of Computer Science, University of Auckland, Private Bag 92019, Auckland 1142, NZ

Christopher J. MacLellan

CMACLELL@CS.CMU.EDU

Human-Computer Interaction Institute, Carnegie Mellon University, Pittsburgh, PA 15213 USA

Abstract

The literature on problem solving in both humans and machines has revealed a diverse set of strategies that operate in different manners. In this paper, we review this great variety of techniques and propose a five-stage framework for problem solving that accounts for this variation in terms of differences in strategic knowledge used at each stage. We describe the framework and its implementation in some detail, including its encoding of problems and their solutions, its representation of domain-level and strategy-level knowledge, and its overall operation. We present evidence of the framework's generality and its ability to support many distinct problem-solving strategies, including one that is novel and interesting. We also report experiments that show the framework's potential for empirical comparisons of different techniques. We conclude by reviewing other work on flexible approaches to problem solving and considering some directions for future research.

1. Introduction

The ability to solve novel problems is one of the hallmarks of human intelligence, and thus a desirable feature of any artificial cognitive system. Not only are people able to construct innovative solutions to tasks they have never before encountered, but their behavior along these lines exhibits two important features. The first is that problem-solving abilities are very general, in that people apply them to a wide range of settings. The second is that humans exhibit substantial variation in their problem-solving strategies, both across different people and across distinct tasks. A full computational theory of problem solving should account for both of these characteristics.

Some early computational studies of problem solving (e.g., Newell et al., 1960) focused on generality, but this emphasis has become far less common in recent decades. Most systems are now specialized to certain classes of problems, such as planning, scheduling, and design, which has led to efficient implementations but also to capabilities that are less general than those found in humans. In addition, most implemented systems adopt a single problem-solving strategy, and in some cases a narrow class of methods dominates an entire subfield.

In this paper, we return to AI's initial concern with generality and address the additional challenge of variability (section 2). We present a theoretical framework that supports generalized problem solving and that accounts for variations in terms of differences in strategic knowledge (section

3). We argue that the theory provides new insights because it not only covers well-known strategies but also suggests novel ones that have not appeared in the literature (section 4.2). The framework also lets us compare the performance of different strategies in a controlled manner, and we illustrate this point with experimental studies that reveal some interesting regularities (section 4.3). In closing, we relate our approach to earlier frameworks – including Soar and PRODIGY – that have addressed variation in problem-solving strategies (section 5) and then consider some avenues for future research (section 6). We note in advance that our aim is not to improve on these theories, but to complement them and explore facets of the topic they did not emphasize. Problem solving involves a broad enough class of phenomena that there seems room for multiple accounts, at least at this stage of our understanding.

2. Aims of the Research

We desire a computational theory of problem solving. A key step in developing any theory is to identify the phenomena one wants it to explain. In this case, the most basic fact is that people are often able to solve nontrivial problems that they have never before encountered. The realization that we might give computers the same ability served as one of the main foundations of the AI revolution in the 1950s, and it remains a major focus of the field. The central role of search in most AI courses and textbooks reflects this abiding concern.

A second phenomena is that the human ability to solve novel problems appears to be very general. Although experts take advantage of domain knowledge to handle familiar types of tasks effectively, they retain the ability to solve, with more effort, unfamiliar problems far outside their areas of expertise. Moreover, problem solving is not limited, as often assumed, to planning tasks that involve generation of action sequences. This ability is also useful in design, scheduling, theorem proving, and solving problems in mathematics and science. We would like a theory that supports such general capabilities.

A third phenomena, more underrated, serves as the primary focus of this paper: humans are highly variable in the strategies they employ to solve problems, and the AI literature on problem solving shows a similar diversity of approaches. For instance, we know that people sometimes chain backward from goals to select operators, as when solving physics problems (Larkin et al., 1980) and when working on puzzles like the Tower of Hanoi (Newell & Simon, 1972). In other cases, such as during chess play, they appear to chain forward from the current state (de Groot, 1978). Similar differences hold between early AI planning systems (Fikes & Nilsson, 1972), which chained backward, and more recent ones (Hoffmann, 2001), most of which chain forward.

Another dimension of variation involves the organization of search. Nearly every AI course teaches students about depth-first, breadth-first, and best-first methods, with the latter being popular in many implemented systems. Humans show less variability here due to their limited short-term memories, relying on progressive deepening (de Groot, 1978) and other variants of greedy search.

Yet another issue, discussed primarily in the AI planning literature, concerns whether one engages in eager or delayed commitment of choices when selecting operators or bindings. There have been no studies of human biases on this front, yet it seems likely that they lean toward eager methods, due to memory limitations, but that they can delay some decisions.

A final dimension involves criteria for terminating search with either failure or success. Different problem-solving systems incorporate different conditions for abandoning branches, such as

detecting loops and exceeding depth limits. They also adopt diverse schemes for determining success, ranging from the costly option of requiring all problem solutions to partial satisfaction with a single solution that achieves only some of the specified goals.

The computational account of problem solving that we develop should account for this variety of strategies, as well as be applicable to a broad range of task domains. Our purpose is not to construct a system that produces better solutions than others or that finds them more rapidly, but rather one that explains the full range of methods observed in human and machine problem solvers. We will not attempt to justify this goal further, as it seems desirable in its own right. However, we will note that such a framework would aid in experimental comparison of different strategies because it would enable their implementation in a common infrastructure, and we report initial studies along these lines in Section 4. In addition, such a theory seems a prerequisite for modeling the adaptive character of problem solving seen in humans, such as their ability to shift between forward and backward search on different problems, and we discuss ideas for extending our framework in this direction in Section 5.

We should note some other topics that are not a focus of our current research. One is that we have not attempted to account for all aspects of human problem solving, such as the influence of limited working memory on strategy choices, even though some clearly make different demands on memory than others. Another is that, following early work in the area, we have focused on problem solving on unfamiliar tasks in which little domain knowledge is available. A third is that we have chosen not to address domains that involve uncertainty. Each area is important, and we hope to explore them in future work, but we must limit our initial scope in order to make progress.

3. A Flexible Theory of Problem Solving

To reiterate, problem solving in humans and machines exhibits both generality and variety, and we want a theoretical framework that covers both of these features. In this section, we present such a framework, dividing our presentation into five parts. We begin by specifying the framework's theoretical assumptions, its representation of problems and solutions, and the architecture it employs to find solutions. After this, we characterize our formalism for encoding the domain knowledge over which the architecture operates. Finally, we describe how the framework encodes strategies that link the architecture with domain knowledge to solve specific problems.

Science often distinguishes between theories and models. In this view, our architectural framework constitutes a theory of problem solving that comprises principles of representation and processing. In contrast, domain knowledge and specific strategies combine to make up models that instantiate this theory to produce behavior. We have implemented both the theory and a variety of strategies in Prolog; we will refer to them collectively as FPS, which stands for *Flexible Problem Solver*, in homage to Newell, Shaw, and Simon's (1960) early work.

3.1 Theoretical Assumptions

Before we present our own theory, we should review the framework for problem solving that has become widely adopted in both AI and cognitive psychology. This standard theory is due originally to Newell, Shaw, and Simon (1958), but it has become so commonly accepted that few now question

whether it has any potential for elaboration and improvement, which is the high-level objective of our research enterprise.

This theory states that problem solving involves carrying out search through a problem space in an effort to transform an initial state into one that satisfies a goal description. This problem space is not enumerated in advance, but rather is generated dynamically by applying operators that transform states into other states. These operators include both conditions on their application and effects they produce when applied. The search process may be guided by heuristics, but it is also organized by strategies that influence the order in which states are considered and in which operators are applied.

Our new theory of problem solving adopts all of these claims, but it also moves beyond them to incorporate some new postulates. These include assumptions that:

- The primary mental structure in problem solving is the *problem*, which includes a state description and a goal description.
- A problem *solution* consists of a problem *P*; an applied operator instance or *intention I*; a *down* subproblem that shares *P*'s state but has goals based on *I*'s conditions; a *right* subproblem that has the same goals as *P* but a state that results from applying *I* to *P*'s state; and solutions to these subproblems. A trivial problem solution is one in which the state satisfies the goals.
- Problems and their (attempted) solutions reside in a *working memory* that changes rapidly over the course of problem solving, whereas operators and strategies reside in a *long-term memory* that changes gradually if at all.
- Problem solving operates in cycles that involve five stages: *problem selection*, *intention generation*, *subproblem generation*, *failure checking*, and *termination checking*. Each stage uses structures in long-term memory to produce changes to problem structures in working memory.
- Long-term memory contains two forms of content: *domain* knowledge that defines predicates and operators for the current problem domain and *strategic* knowledge that specifies the problem-solving strategies used at each of the five stages.

Although the first three assumptions specify important commitments about representation and organization, the final two tenets are the most interesting and important. We discuss them later in the section, after clarifying some issues of representation and organization.

3.2 Structure of Problems and Solutions

Before we discuss our framework's mechanisms for problem solving, we should first consider its representational assumptions. As noted, the primary structure is the *problem*, which always has an associated state description and a separate goal description, each with its own identifier so that it can be reused elsewhere. A nontrivial problem (one for which the state does not satisfy the goals) has zero or more operator instances or *intentions*, each of which decomposes it into a down subproblem and a right subproblem. A decomposition of problem *P* is a *solution* to *P* if its down subproblem and right subproblem themselves have solutions, with trivial subproblems serving as terminal nodes.

We can clarify this organization with an example from the Tower of Hanoi domain. Recall that this involves moving disks to target pegs subject to the constraints of moving only one disk at a time, not moving a disk if a smaller one is on it, and not moving a disk to a peg if a smaller one is

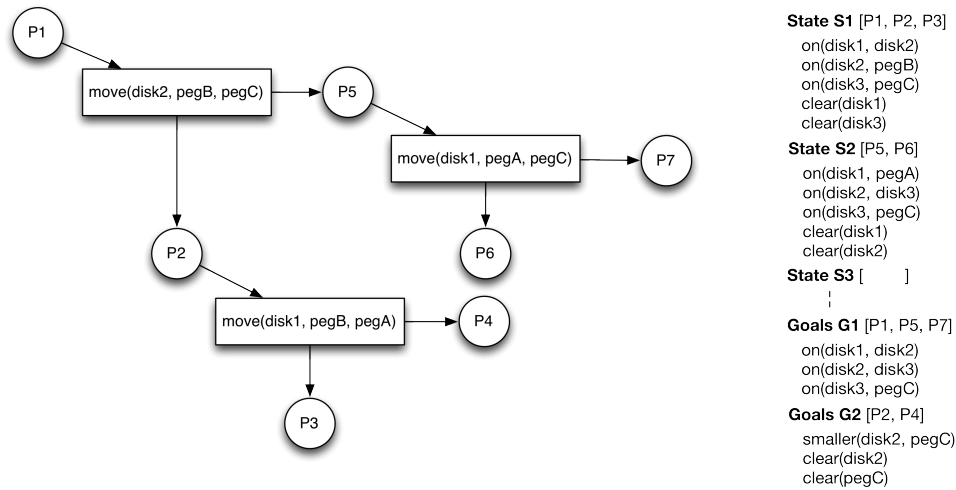


Figure 1. Solution decomposition for a simple Tower of Hanoi problem (P1) that includes both down-branching (P2) and right-branching (P5) subproblems. Each terminal node denotes a trivial subproblem.

already there. Figure 1 shows the structure of a solution for a simple task that involves moving disks 1 and 2 from peg B onto a target peg C that already holds disk 3. The simplicity of this problem (P1) is reflected in the structure of its solution. This is not the only possible solution structure for this problem, but it is a reasonable one that lets us clarify the organization of elements in memory.

This involves a decomposition of P1 into a down subproblem (P2), since its intention – moving disk 2 from peg B to peg C – is not applicable in state S1, and a right subproblem (P5), since this operator instance does not by itself achieve all of P1’s goals (G1). The down subproblem P2 is decomposed further based on another intention – moving disk 1 from peg B to peg A – that produces two additional subproblems (P3 and P4). These are both trivial, in that the intention is applicable in S1 and achieves all of the goals (G2) associated with P2. The conditions for the original intention (moving disk B) match the resulting state S2, with a new state (S3) produced by its application that is associated with right subproblem P5. The intention attached to P5 – moving disk 1 from peg A to peg C – has trivial subproblems, as its conditions match S3 and it achieves the remaining goal in G1, thus solving the top-level problem.

This example is simple, but it should clarify the nature of problem decompositions and the organization of solutions, which is inherently hierarchical. We maintain that other frameworks are special cases of this general idea. For instance, the popular class of methods that approach problem solving in terms of forward search are techniques that only consider solutions with nontrivial right-branching decompositions. In contrast, the early Logic Theory Machine (Newell, Shaw, & Simon, 1957) only considered solutions with nontrivial down-branching decompositions. Our framework’s support for both types of solution borrows from the decompositional scheme introduced in Newell, Shaw, and Simon’s (1960) General Problem Solver. Although their implementation had many draw-

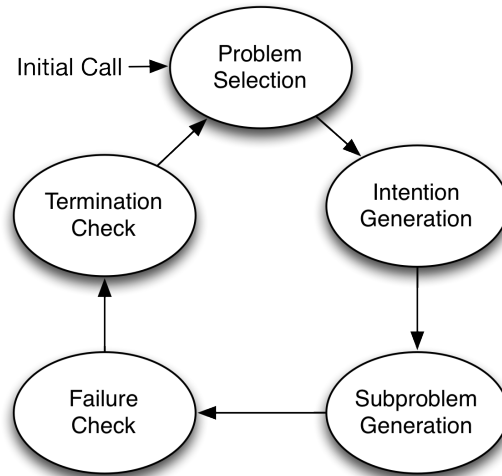


Figure 2. The five-stage cycle of the FPS problem-solving architecture.

backs, we hold that its representation of problem decompositions is ideally suited to support the variety of search strategies that we aim to explain.¹

3.3 Five Stages of Problem Solving

As just noted, our problem-solving architecture cycles repeatedly through five stages. The first step involves selecting a problem on which to focus attention. There is only one problem at the outset, but the number of choices increases on later rounds. During this stage, the problem solver examines working memory to determine which problems are available (i.e., not marked as solved or failed) and alters it to reflect the newly selected focus. Different strategic knowledge for this stage imposes different organizations on search through the problem space. For example, selecting more recently created problems leads to strategies like depth-first search and iterative deepening (Langley, 1992), whereas selecting less recent ones produces ones like breadth-first and beam search.

Once the architecture has selected a problem P , the second stage selects an operator instance that it hopes will lead to a solution for P . We refer to these as *intentions* because the problem solver intends them for execution if they participate in its final solution for P . This stage examines both P and domain knowledge about operators, considers all intentions that are available and relevant, selects one of them, and associates it with P . If no operator instances are available (e.g., if all relevant candidates have failed on earlier attempts), the problem solver annotates the problem to that effect. Different strategic knowledge for this stage generates intentions by chaining backward from problem goals, chaining forward from the state, or some mixture of these schemes.

The third stage generates new subproblems of the current problem P , if necessary, based on the newly selected intention I , if one has been produced. This includes creating a down subproblem

1. The related technique of *island search* (Chakrabarti, Ghose, & Desarkar, 1986) also decomposes a problem into subproblems, but our approach comes closer to that in Newell et al.'s earlier work.

Table 1. Example FPS encodings for a state description, a goal description, and an operator for the Tower of Hanoi puzzle, with each being described as a set of literals.

<p><i>State description:</i> state(s1, on(disk1, disk2)), state(s1, on(disk2, disk3)), state(s1, on(disk3, pegA)), state(s1, clear(disk1)), state(s1, clear(pegB)), state(s1, clear(pegC)).</p> <p><i>Goal description:</i> goal(g1, on(disk1, Any_disk)), goal(g1, on(disk3, pegC)), goal(g1, on(Any_disk, disk3)).</p>	<p><i>Operator description:</i> operator(move(Disk, From, To)), condition(move(Disk, From, To), smaller(Disk, To), condition(move(Disk, From, To), clear(Disk)), condition(move(Disk, From, To), clear(To)), effect(move(Disk, From, To), on(Disk, To)), effect(move(Disk, From, To), clear(From)).</p>
---	---

that has the same state as P but has goals based on I 's conditions, along with a right subproblem that has the same goals as P but has a state produced by applying I 's effects to P 's state.² Different strategies at this stage include eager commitment, which creates subproblems as soon as an intention is available, and delayed commitment, which waits until a set of intentions have been found, then determines the order in which they should be applied.

Another stage checks for failures that indicate the problem solver should abandon the current problem P . This involves inspecting P and its associated intentions for unresolvable issues and adding relevant annotations to P about them. For example, this stage can check for an absence of available intentions, loops in the path that led to P , evidence that P is more difficult than one of its ancestor problems, reaching a depth limit, or even that too many attempts have been made to solve it. Such tests are not usually given the status of problem-solving strategies, but they are just as important as constructive steps that create intentions and subproblems.

The fifth and final stage checks to see whether any additional work is needed to solve the current problem P . This compares P 's state and goal descriptions to determine whether there is a sufficient match. If so, then it annotates P as solved, which will influence problem selection on the next cycle. Different strategic knowledge for this stage specifies alternative criteria for deciding when a problem is solved. This may require that all goal elements are matched, that some percentage of goals are satisfied, or, in settings where goals have associated values, that the summed value of matched goals exceeds a threshold.

To summarize, our problem-solving framework incorporates five stages: problem selection, intention generation, subproblem creation, failure checking, and termination checking. The architecture cycles through these steps, combining strategic knowledge associated with each stage and domain knowledge about operators to carry out search through the problem space that they jointly define. To understand how each stage operates, we must examine these knowledge elements, to which we now turn our attention.

2. The resulting down subproblem will be trivial if I 's conditions already match the current state, while the new right subproblem will be trivial if the result satisfies all of P 's goals.

3.4 Domain Knowledge for Problem Solving

According to the standard theory, problem solving depends on domain operators to generate new states that achieve goal descriptions, which in turn requires some way to represent all three types of structures. Our implementation of the FPS architecture adopts a logic-like notation for states and goal descriptions, in which each is specified by a set of literals (domain predicates with arguments). Goal descriptions may differ from states by omitting some elements, using variables rather than constants for some arguments, and including negated literals. Our notation for operators differs from traditional ones like STRIPS and PDDL, encoding each operator as a *set* of generalized literals, which supports more flexible processing by the problem-solving architecture.³

Table 1 shows the representation for a state description and a goal description from the Tower of Hanoi, along with that for the single operator for this domain. Each state literal describes a different aspect of the state, while the same holds for the more abstract goal description, which contains the pattern-match variable *Any_disk* in two elements. The operator description includes a literal that names the operator and its arguments, three condition elements that together specify when the operator is applicable and how their arguments relate, and two effect elements that jointly specify how the operator changes the state description. Although this example does not illustrate them, conditions may be negated and effects may involve deletions.

3.5 Strategic Knowledge for Problem Solving

Of course, domain knowledge by itself is not sufficient; we need some way to interpret the domain states, goal descriptions, and operators to produce problem solving. In most AI research, this takes the form of opaque, procedural code that, although efficient, lacks the flexibility needed to support a variety of strategies.⁴ Instead, our framework specifies problem-solving behavior in terms of domain-independent knowledge.

The details of this strategic knowledge differ for each of the architecture's five stages, but they share some features that distinguish them from domain knowledge. In particular, they:

- make no reference to either domain-level predicates or operators, containing instead variables that match against such domain content;
- refer to meta-level predicates like *problem*, *state*, *goal*, *operator*, *condition*, *effect*, and *intention*;
- consist mainly of control rules that specify decisions to make under various conditions; and
- include inference rules that support control rules by determining whether their conditions hold.

The control rules for a given stage *S* are stored in an ordered list that, upon entering *S*, the architecture considers in turn, invoking inference rules as needed to determine whether a given control rule's conditions are satisfied. If so, then the rule fires and alters the contents of working memory. We can clarify this process by discussing briefly the control schemes that we have implemented within the framework.

3. We are not claiming that our formalism has greater expressive power than traditional ones, only that its distributed character offers benefits for encoding and processing strategic knowledge.

4. We do not mean that procedural encodings cannot, in principle, offer flexibility, say through parameters or switches, but they have been associated empirically with inflexible approaches to problem solving.

Recall that the first stage involves selecting a problem on which to work. Here we have implemented three alternative regimes: depth-first search (two control rules), iterative sampling (two rules), and breadth-first search (two rules). Table 2 (a) presents the control rules used to produce depth-first search. Iterative sampling, which carries out repeated greedy search, replaces one of these rules but uses the other one, which provides evidence for the modularity of this knowledge.

The second stage is responsible for generating intentions (operator instances) for use on the current problem. In this case, we have implemented two approaches: backward chaining, which considers an operator only if its application would achieve one or more problem goals, and forward chaining, which considers an operator only if all its conditions match the problem state. Table 2 (b) presents the two control rules used for backward chaining; the forward version uses the same number of rules. The architecture also ranks intentions generated through both mechanisms by the number of goals they achieve. Informal studies suggested that these variants guided search substantially better than ranking candidates equally.

The next stage selects the most highly ranked intention, determines whether to proceed with this choice, and, if so, generates its associated down and right subproblems.⁵ Here we have implemented two alternatives that involve eager and delayed commitment. The former utilizes four control rules, whereas the latter requires twelve. Delayed commitment also includes a substantial number of inference rules that detect interactions and dependencies (such as goal clobbering) among the candidate intentions and thus modulate their final ordering.

The framework's final two stages handle failure and termination. Control knowledge implemented for failure comprises nine rules, including ones that match when an ancestor problem is the same or more difficult than the current one, when all decompositions have been attempted, and when a depth limit has been exceeded. Analogous control knowledge for termination includes four rules that apply when the problem's state matches its goal description, when a right subproblem has been labeled as solved, when attempts to solve the top-level problem have failed, and when this problem has been solved.

Taken together, the control and inference rules for each stage specify a complete problem-solving strategy. Combined with the architecture, which cycles repeatedly through the stages, and the domain operators and predicates, they let the system attempt to solve novel tasks through a process of problem-space search. Each stage can utilize alternative schemes to organize and direct this search. Differences in control knowledge for each stage produces the variability we have argued is characteristic of human problem solving and desirable in cognitive systems generally.

4. Experience with the Framework

Now that we have described our problem-solving framework and its implementation, we can turn to whether it satisfies the aims we outlined early in the paper. In this section, we examine the architecture's functionality, its generality, and its ability to support a wide variety of strategies, one of which is novel and interesting. We also report experiments that demonstrate the framework's support for empirical studies of alternative problem-solving techniques.

5. When two or more intentions have the same ranking, the architecture selects among the candidates at random.

Table 2. Control rules used to implement (a) depth-first search and (b) backward chaining.

(a) Depth-first selection of problems

If no problem is currently selected, then select the root as the current problem.

If there currently is a selected problem, then select as the new current problem the one with the highest identifier that has not been solved and that has not failed.

(b) Backward-chaining generation of intentions

If the current problem P has already been labelled as having no operators available, then do nothing.

If an operator instance I exists that achieves one or more goals for current problem P , and I is not already an intention for problem P , then make I an intention of P .

4.1 Basic Functionality and Generality

The most basic claims about our framework are that it supports effective problem solving on novel tasks and that it exhibits generality by solving problems across a wide range of domains. To demonstrate that it satisfies these criteria, we have implemented in Prolog (Clocksin & Mellish, 1981) the problem-solving architecture, a variety of problem-solving strategies that we will discuss shortly, and domain knowledge for various domains.

Table 3 lists the nine different domains for which we have developed predicates and operators. These include classic puzzles like Tiles and Squares (Ohlsson, 1982), the Tower of Hanoi (Newell & Simon, 1972), Missionaries and Cannibals, Slide Jump, and the Blocks World (Fikes & Nilsson, 1972), planning domains like Gripper, Logistics, and Dock Workers, and inference tasks like deducing kinship relations. These differ substantially in their characteristics. For instance, the three puzzles involve simple movement or stacking of objects subject to constraints, whereas the three planning domains incorporate more complex forms of action. These domains all involve nonmonotonic operators that can make existing relations false, whereas those for inferring kinship relations are entirely monotonic.

The fact that FPS can solve problems in each of these domains does not prove its generality, but it certainly lends evidence to that claim. However, many previous implementations, starting with Newell, Shaw, and Simon's GPS and continuing through many planning systems, have shown similar generality and have been tested more thoroughly. Thus, we now shift attention to our framework's more distinctive capabilities.

4.2 Coverage of Existing and Novel Strategies

Another primary claim is that our theoretical framework has broad coverage of the problem-solving strategies exhibited by both human and artificial problem solvers. By coverage, we mean that the architecture is capable of reproducing a wide range of strategies from the literature.⁶ For example, if the framework could only reproduce the behavior of a single system, its coverage would be

6. Naturally, the size of this space is difficult to quantify in advance, but we will be able to tell, as our theory progresses, whether its coverage of known methods is increasing.

Table 3. Problem-solving domains on which we have tested the FPS architecture.

Tiles and Squares. This domain involves a set of N tiles that sit on $N + 1$ squares, so that one square is always empty. The single operator can move any tile from its current square onto the empty square, with the goal being to rearrange the initial configuration of tiles into a target arrangement.

Blocks World. This domain consists of a number of separate blocks and a table. The four operators let one pick up a block from the table or off another block and deposit a block onto the table or another block. Goal descriptions involve partially specified configurations of block and table relations.

Tower of Hanoi. This domain involves N disks that sit on three pegs. The single operator can move a disk onto a new peg if there is no smaller disk on it and if there is no smaller disk on the new peg. Goal descriptions specify desired placements of disks on pegs.

Missionaries and Cannibals. This domain concerns a boat, two river banks, N missionaries, and N cannibals. All missionaries and cannibals begin on one bank and must be transported to the other side, but there must never be more cannibals than missionaries on either bank. The single operator moves the boat from one bank to another, along with one to three passengers.

Slide Jump. In this domain, N dimes and N nickels are arranged in a row of $2N + 1$ locations, one of which is empty. Initially, all dimes are left of the empty slot and all nickels are right of it. One operator slides a dime right into an adjacent empty slot or slides a nickel left into one; the other jumps a dime right over a nickel into an empty slot or jumps a nickel left over a dime. The goal is to get all dimes to the right of the empty slot and all nickels to its left.

Kinship Relations. This domain involves people who are in parent-child relationships, with monotonic operators for inferring more complicated forms of kinship that appear in goal descriptions.

Gripper. This domain involves a robot with a gripper, a number of balls, and two or more rooms. Operators include gripping a ball, moving to a different room, and dropping a ball, with goals specifying desired locations of balls.

Logistics. This domains includes a number of cities, locations in those cities, transportable packages, and trucks and plans that can hold those packages. Operators involve loading and unloading packages, flying planes between cities, and driving trucks between locations. Goals specify desired locations of packages.

Dock Workers. This domain contains a robot, containers, pallets, and cranes in various locations. Operators let the robot move between locations and let the crane transfer a container from the top of a stack to the robot or vice versa. Goal descriptions specify desired positions of containers on pallets in locations.

very poor, but if it could reproduce the behavior of a number of diverse systems, then its coverage would be high. This metric is both relevant and important because it shows how effectively our theoretical framework explains the phenomena in question, in this case the variety of problem-solving strategies. To evaluate the coverage of our system, we will describe how it reproduces several problem-solving strategies from the literature and show how strategies that are not supported by our current models are still supported by the theoretical framework.

We have implemented some 12 distinctive problem-solving strategies that result from different choices of control knowledge for the five stages. These include some familiar techniques that are often presented as standalone problem-solving methods in textbooks:

- Breadth-first forward search. This strategy results from a combination of breadth-first problem selection, generation of intentions by forward chaining, eager commitment, loop-triggered failure, and success upon finding a single state that matches the goal description. Because this strategy only considers operators with matched conditions, it produces entirely right-branching paths that fan out from the generated states.
- Depth-first means-ends analysis. This strategy (Newell, Shaw, & Simon, 1960; Fikes & Nilsson, 1972) results from a combination of depth-first problem selection, generation of intentions by backward chaining, eager commitment, loop-triggered failure, and success upon finding a single state that matches the goal description. Because this strategy considers operators that achieve one or more goals but may not have the conditions satisfied, it typically generates solution paths with a mixture of down and right subproblems.
- Forward search with iterative sampling. This strategy (Langley, 1992) combines iterative sampling – repeated greedy search with no memory of previous passes – to a fixed depth with forward chaining to generate intentions, including eager commitment and success upon finding a single solution. This is another strategy that only considers operators with matched conditions, so it also generates paths with only right subproblems.
- Depth-first partial-order planning. This strategy combines depth-first problem selection with backward generation of intentions and delayed commitment for subproblem generation. Although this scheme typically produces solutions with a mixture of down and right subproblems, it checks for order dependencies among intentions before selecting one, applying it mentally, and creating right subproblems.

We have focused on these examples because they have been described in the literature and they will be familiar to many readers. However, a more interesting question is whether the framework suggests novel and interesting combinations of settings that have not appeared to date.

An important feature of our framework is that its modularity lets us combine any substrategies for one stage with any substrategies for another. This is what lets us produce so many distinct problem-solving strategies given the elements discussed in Section 3. Many of these are minor variations of each other, such as a depth-first version of forward search and a breadth-first variant of partial-order planning, while others correspond to techniques that have appeared in the literature but are not well known, such as the union of backward chaining with iterative sampling (Jones & Langley, 2005).

However, at least one strategy that emerges in this manner, and which is unexpected and interesting, involves a combination of least commitment with forward chaining. Least commitment was originally designed for use with backward-chaining methods, specifically ones for partial-order planning, but our framework also allows its use in conjunction with forward search. In this strategy, the problem solver generates a number of intentions with matched conditions, but then examines them for interactions before selecting one to drive creation of subproblems. To our knowledge, this

idea has never been reported in the AI or cognitive psychology literature and, as we will see shortly, this scheme produces interesting and unexpected behavior.

This result provides evidence that our problem-solving framework does more than cover already established strategies; it suggests novel combinations that have never been observed. In this sense, it is playing a similar role to Mendeleev’s periodic chart, which not only organized known elements into a coherent and structured framework, but also suggested new elements that would fill gaps in the table. The fact that our theoretical framework and its associated implementation supports such generality is an encouraging sign, although it seems clear that some strategies remain outside its current incarnation and there remains considerable room for improvement.

4.3 Experimental Studies of Alternative Strategies

One advantage of our framework’s implementation is that it lets us compare the effectiveness of different strategies experimentally. Most modern problem-solving systems (e.g., Hoffmann, 2001) rely on a single, highly optimized strategy, so that differences in their behavior may be due either to implementation details or to more basic distinctions. Because instances of FPS rely on the same underlying architecture and differ only in their strategic control rules, we can eliminate variance due to other factors and thus carry out more direct and revealing empirical comparisons among alternative problem-solving strategies.

Figure 3 (a) presents experimental results from FPS runs on 23 distinct problems collected for seven of the task domains described earlier in Table 3: Blocks World, Tower of Hanoi, Missionaries and Cannibals, Slide Jump, Kinship, Logistics, and Dock Workers. We ran FPS using each of the 12 strategies on three problems from each domain except for Logistics and Blocks World, for which we used two and six problems, respectively. Most of the problems in a given domain had similar complexities in terms of number of objects and length of solution, but, in many cases, each task was solved more easily with some strategies than others.

The graph plots the CPU times for all six of the forward-chaining FPS variants against all six backward-chaining versions. Each point compares the performance on a given problem of one forward-chaining strategy with its backward-chaining analog, holding settings for other FPS stages (e.g., problem selection or subproblem generation) constant. For problems that fall above the diagonal, forward chaining generation of intentions took longer to find a solution than did backward chaining, with the reverse holding for problems below this line.⁷ One clear trend is that backward-chaining strategies fare better on the kinship domain; this result is unsurprising, as the operators are monotonic and thus lend themselves to goal-directed processing.

Figure 3 (b) presents the results from a more focused comparison between the novel class of strategies described earlier – the combination of forward chaining with delayed commitment – with the average of all other strategies. In this case, problems above the diagonal denote those on which one strategy class took less time than the other class. The graph shows that, except for some tasks in the upper right corner that are difficult in general, the new class of strategies fares better than the average of all other methods. Detailed analysis suggests that this combination is effective because it avoids the cost of checking for interactions on down subproblems, since these are trivial with

7. We halted runs after 3,000 problem-solving cycles, so the cluster of points at the far right means that backward-chaining techniques exceeded this bound on many tasks.

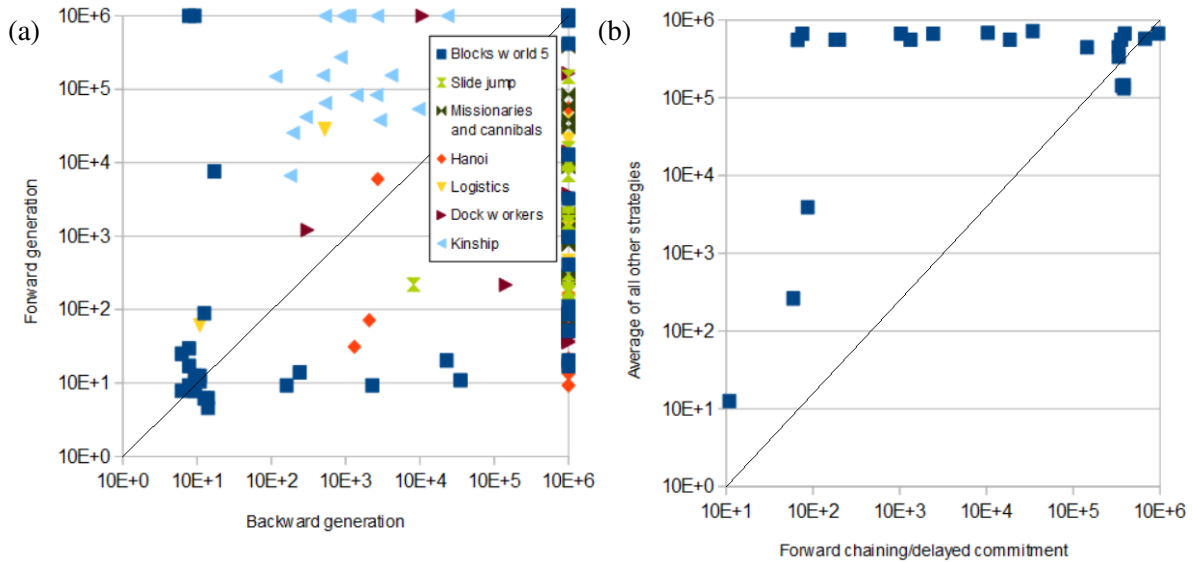


Figure 3. Scatter plots that compare the CPU seconds needed for FPS to solve problems in five domains (a) comparing variations on forward chaining with variants of backward chaining, as well as (b) comparing forward chaining, delayed commitment variants against averages of all other strategies.

forward chaining, and reaps the benefits of delayed commitment when it comes close to achieving the goal description, when interactions among operators often become important, without paying much overhead early in the search process.

These experimental results are intriguing and suggest the need for additional studies of forward chaining with delayed commitment, but they are not, in themselves, the main point of our research. The conclusion readers should draw is that this comparison would not have occurred without our framework’s capacity for suggesting and supporting novel problem-solving strategies, as well as its ability to compare such search methods experimentally in a controlled manner. The results provide evidence that the framework enables and fosters both forms of scientific activity.

5. Related Research

Problem solving has played an important role in AI’s history, and there is a substantial body of work from which we have drawn. Although early efforts (e.g., Newell et al., 1960) emphasized generality more than recent ones, even today’s specialized planning systems exhibit substantial coverage of many tasks. Thus, we will not claim that our framework’s generality is especially novel, despite its applicability to non-planning problems, such as inference tasks that involve monotonic operators.

However, the issue of variability is another matter entirely, as most research paradigms are now dominated by a single approach, such as forward search in planning and backward search in deductive inference. To find exceptions to this trend, we must examine the older literature, before the

modern emphasis on systems that are optimized for CPU time. One example comes from Kambhampati and Srivastava (1995), whose framework for “universal classical planning” attempted to unify “plan-space” and “state-space” planning by casting each as special cases of refining a partial order plan. Their framework supported three distinct strategies: forward search through a state space, backward search through such a space, and mixed search through a plan space. Although similar in spirit, our framework provides much broader coverage of strategies.

Another early effort that supported a broad range of problem-solving behaviors revolved around the PRODIGY architecture (Carbonell, Knoblock, & Minton, 1990). This framework is quite similar to ours in that it incorporated a decision-making loop that, sequentially, selects among goals, states, operators, and bindings. These processing steps do not map precisely onto our five stages, but they play a similar role. PRODIGY also specified alternative strategies in terms of control rules, although an important difference is that published work on this framework emphasized using and acquiring domain-specific control knowledge, whereas we are concerned with domain-independent strategies. Minton (personal communication, 2012) reports that PRODIGY could utilize domain-independent control rules, but the project did not explore this ability systematically.

The FLECS system (Velooso & Stone, 1995), which extended the PRODIGY framework, came even closer to our approach by supporting both eager and delayed commitment, a capability handled by our third stage. FLECS could also shift between progression and regression approaches to planning, which maps directly onto alternatives in our architecture’s second stage of intention generation. However, our framework also supports different schemes for problem selection, failure, and termination. Also, like PRODIGY, it only considered operators that would achieve at least one of the current problem’s goals, so it could not produce purely forward search. Thus, we can view our FPS framework as an extended, more flexible version of FLECS that covers a wider range of problem-solving behaviors.

A fourth theoretical framework with similar aims is the Soar architecture (Laird et al., 1987). This organizes behavior around problem-space search in a six-stage cycle that checks for success, decides on failure or suspension, selects a state, selects an operator, applies the operator, and decides whether to save the resulting state. Each stage examines control rules that vote for or against available alternatives, complemented by an elaboration process that applies inference rules to inform these decisions. Although recent research on Soar has emphasized domain-specific control knowledge, early work utilized this approach to mimic a variety of “weak methods” that embodied generic problem-solving techniques. These correspond to our settings for various stages, such as depth-first search and backward chaining, and they could be combined in much the same way as FPS does. Our framework supports dimensions of variation not (to our knowledge) studied in Soar, such as eager and delayed commitment, but the two theories still have much in common.

These earlier efforts may lead some readers to question why we have developed our own framework for flexible problem solving. One reason is that, despite the early PRODIGY and Soar results, the topic has received remarkably little attention for over two decades, and it deserves more air time. Another is that FPS includes some new emphases, such as the hierarchical character of problem solutions and additional dimensions of variation. We will not claim that our theory is superior to its predecessors, but we believe that the field’s understanding of problem is still far from complete, and

that achieving human-level flexibility in this arena is challenging enough to justify more alternative accounts of this important set of phenomena.

6. Limitations and Responses

Despite our progress to date, there remain many directions in which we can extend both our theoretical framework and its implementation. The first step should be to support an even wider range of strategies in FPS by adding new control knowledge for various stages, especially problem selection, which should include techniques like beam search, best-first search, and iterative deepening. We should also add the ability to utilize numeric evaluation functions, both generic and domain-specific ones, to guide selection of both problems and intentions.

In addition, we should demonstrate that the framework supports other classes of problem-solving tasks. For instance, more flexible termination criteria should produce techniques for partial satisfaction planning (van den Briel et al., 2004), which find plans that achieve only a subset of the specified goals. The framework should also handle optimization techniques that do not halt at one solution but continue looking for alternatives that fare better on some evaluation function. Constraint satisfaction tasks should also be straightforward, requiring only that some goals be stated in terms of derived predicates. Naturally, we should continue to look for opportunities in which the fine-grained character of our control rules suggest novel strategies that support viable problem solving, in which case we should compare them experimentally to established techniques.

Of course, we should also explore extensions to the problem-solving architecture itself. One high priority is to support the use of hierarchical task networks ((Nau et al., 2001) to organize and constrain the search process. The decompositions that FPS already uses during problem solving are similar in spirit to HTN methods. We intend to store generalized versions of these decompositions, with associated conditions and effects, as high-level operators on which the system can draw during search. A heuristic that prefers intentions which achieve more goals should favor hierarchical methods over primitive operators, which would still be available as fallback options. The mapping between specific problem decompositions and HTN methods also suggests the system can learn the latter from successful problem solutions.

Yet another promising avenue involves shifting between strategies dynamically, based on measures like relative branching factors in the forward and backward directions. A different augmentation would introduce stages for reformulating problems when they prove difficult to solve (MacLellan, 2011; Riddle, 1990) and for translating solutions in the reformulated space back into the original one. This will require specifying reformulation operators that alter the representations for states, goals, or operators. An even more radical extension would provide the framework with an ability to generate new control rules and thus explore the space of strategies automatically. Each of these extensions would involve incremental additions to the architecture rather than a major revision, which further suggests the benefits of our modular framework.

7. Concluding Remarks

In this paper, we noted that, although the literature on problem solving in humans and machines describes many approaches, most implemented systems adopt a single, unvarying strategy. In response, we presented a theoretical framework that accounted for this variation as differences in strategic knowledge that is interpreted, along with domain knowledge, by a problem-solving architecture. The architecture decomposes each problem into an intention, a down subproblem, and a right subproblem, and it operates in five stages that focus, in turn, on problem selection, intention generation, subproblem creation, failure checking, and termination checking.

We demonstrated that an implementation of this framework solves problems in seven distinct domains and that it supports 12 major problem-solving strategies, along with many minor variations. We saw that some of these map directly onto familiar techniques that have appeared widely in the literature, with most others being minor variations on them. However, we also showed that one strategy – combining forward chaining with least commitment – was novel and interesting, showing that our framework has the power to predict new techniques. We also carried out experiments with the various strategies, finding that the new scheme fared surprising well compared to more standard methods and showing the framework’s support for empirical studies.

We examined a number of earlier architectures that have supported flexible approaches to problem solving, noting that our framework is similar in spirit to these precursors but that it addresses some distinctive issues. Finally, we acknowledged that both our framework and the FPS system remain limited in their coverage, and we outlined some additions that should improve their explanatory range. Our architecture already reproduces a wide variety of search behaviors, but such extensions would extend its account of problem solving substantially.

Acknowledgements

This research was supported by Grant No. N00014-10-1-0487 from the Office of Naval Research. We thank Jaime Carbonell, Steve Minton, Peter Stone, Manuela Veloso, and Subbarao Kambhampati for providing information about earlier work in the same tradition, as well as Colin Walker and Trevor Gee for their efforts on earlier versions of the system.

References

- Carbonell, J. G., Knoblock, C. A., & Minton, S. (1990). PRODIGY: An integrated architecture for planning and learning. In K. Van Lehn (Ed.), *Architectures for intelligence*. Hillsdale, NJ: Lawrence Erlbaum.
- Chakrabarti, P. P., Ghose, S., & Desarkar, S. C. (1986). Heuristic search through islands. *Artificial Intelligence*, 29, 339–348.
- Clocksin, W. F., & Mellish, C. S. (1981). *Programming in Prolog*. Berlin: Springer-Verlag.
- de Groot, A. D. (1978). *Thought and choice in chess* (2nd Ed.). The Hague: Mouton Publishers.
- Fikes, R. E., & Nilsson, N. J. (1972). STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2, 189–208.
- Hoffmann, J. (2001). FF: The Fast-Forward planning system. *AI Magazine*, 22, 57–62.

- Jones, R. M., & Langley, P. (2005). A constrained architecture for learning and problem solving. *Computational Intelligence*, 21, 480–502.
- Kambhampati, S., & Srivastava, B. (1995). Universal Classical Planner: An algorithm for unifying state-space and plan-space planning. *Proceedings of the Third European Workshop on Planning Systems*. Assisi, Italy.
- Laird, J. E., Newell, A., & Rosenbloom, P. S. (1987). Soar: An architecture for general intelligence. *Artificial Intelligence*, 33, 1–64.
- Langley, P. (1992). Systematic and nonsystematic search strategies. *Proceedings of the First International Conference on Artificial Intelligence Planning Systems* (pp. 145–152). College Park, MD: Morgan Kaufmann.
- Larkin, J. H., McDermott, J., Simon, D. P., & Simon, H. A. (1980). Expert and novice performance in solving physics problems. *Science*, 208, 1335–1342.
- MacLellan, C. (2011). An elaboration account of insight. *Proceedings of the 2011 AAAI Fall Symposium on the Advances in Cognitive Systems*. Arlington, VA: AAAI Press.
- Nau, D. S., Cao, Y., Lotem, A., & Muñoz-Avila, A. (2001). The SHOP planning system. *AI Magazine*, 22, 91–94.
- Newell, A., Shaw, J. C., & Simon, H. A. (1957). Empirical explorations of the Logic Theory Machine. A case study in heuristic. *Proceedings of the Western Joint Computer Conference* (pp. 218–230) New York: Institute of Radio Engineers.
- Newell, A., Shaw, J. C., & Simon, H. A. (1958). Elements of a theory of human problem solving. *Psychological Review*, 65, 151–166.
- Newell, A., Shaw, J. C., & Simon, H. A. (1960). Report on a general problem-solving program for a computer. *Proceedings of the International Conference on Information Processing* (pp. 256–264). UNESCO House, Paris.
- Newell, A., & Simon, H. A. (1972). *Human problem solving* Englewood Cliffs, NJ: Prentice-Hall.
- Ohlsson, S. (1982). On the automated learning of problem solving rules. *Proceedings of the Sixth European Meeting on Cybernetics and Systems Research*.
- Penberthy, S. J., & Weld, D. S. (1992). UCPOP: A sound, complete, partial order planner for ADL. *Proceedings of the third international conference on knowledge representation and reasoning* (pp. 103–114). Cambridge, MA: Morgan Kaufmann.
- Riddle, P. J. (1990). Automating problem reformulation. In D. P. Benjamin (Ed.), *Change of representation and inductive bias*. Boston: Kluwer Academic Publishers.
- van den Briel, M., Nigenda, R. S., Do, M. B., & Kambhampati, S. (2004). Effective approaches for partial satisfaction (over-subscription) planning. *Proceedings of the Nineteenth National Conference on Artificial Intelligence*. San Jose: AAAI Press.
- Veloso, M., & Stone, P. (1995). FLECS: Planning with a flexible commitment strategy. *Journal of Artificial Intelligence Research*, 3, 25–52.

Preliminary Results on a Meta-Level Search Framework for Problem Reformulation

Patricia J Riddle

PAT@CS.AUCKLAND.AC.NZ

Michael W Barley

BARLEY@CS.AUCKLAND.AC.NZ

Santiago M. Franco

SANTIAGO.FRANCO@GMAIL.COM

Private Bag 92019, Department of Computer Science, University of Auckland, Auckland, 1142 NZ

Abstract

A problem's representation has a big impact on how hard a problem is to solve, ranging from easy to solve to intractable. We describe a framework for meta-level search through a space of problem representations. These new problem representations are not predefined but are generated using a set of transformations. We describe a case study of a single composite transformation. Using this composite transformation, we automatically transform the PDDL representation of the GRIPPER domain into a new PDDL representation. In this new representation, the previously intractable GRIPPER problems can now be solved in a few seconds.

1. Introduction

A problem's representation can make it either easier or harder to solve a problem. This is true for both humans and for computers (Newell, 1966; Newell & Simon, 1972; Hayes & Simon, 1974). A natural response for a human when encountering a difficult problem, is to change the problem representation. For instance, Polya states "Trying to find the solution, we may repeatedly change our point of view, our way of looking at the problem. We have to shift our position again and again. Our conception of the problem is likely to be rather incomplete when we start the work; our outlook is different when we have made some progress; it is again different when we have almost obtained the solution." (Pólya, 1957) Our framework allows an automated problem solver to exhibit the same behavior, and change the problem representation in hopes of making the problem easier to solve.

It has been shown by many researchers since the 1970s that the ability to change a problem representation to a new (and hopefully easier to solve) representation is a key strategy that people use, and that this would be a key asset for a true artificially intelligent system (Amarel, 1968; Amarel, 1971; Simon, 1972). In this paper we give a framework for addressing this challenge. Our approach is a meta-level search through a space of problem representations. These new problem representations are not predefined but are generated using a set of transformations. To accomplish this we need a language for representing the problem representations and we need a set of transformations for moving between them. Given that we have generated a new problem representation, a problem solver can search the space defined by this new representation. If necessary, solutions in the new problem representation can be transformed back into solutions in the original representation.

Our claims are that this powerful framework enables flexible problem solving, by supporting automatic transformations from intractable problem representations to easier problem representations. This allows a problem solver to solve new problems that it could not solve before, in reasonable time and space. We have previously define a set of component transformations, see (Riddle, Barley, & Franco, 2013), out of which higher-level composite transformations can be defined. We explore a case study of a single composite transformation. We automatically transform the PDDL description of the GRIPPER domain into a new PDDL description, where previously intractable problems can now be solved in a few seconds.

The next section discusses the standard theory of problem reformulation. In the following sections, we discuss representing problem representations, transformations for problem reformulation, and a framework for problem reformulation. We then explore the case study of a composite transformation providing encouraging results in the GRIPPER domain. After this we conclude and offer directions for future research.

2. Standard Theory of Problem Reformulation

We will briefly review the work adopted in our framework. Early work from Gestalt psychology talks about reformulation on insight problems. A good survey of this work is Ohlsson (1992). The early research in the area of problem reformulation in Artificial Intelligence (AI) started in the 1960s (Amarel, 1968; Amarel, 1971; Simon, 1972; Korf, 1980).

Reformulation is a general term meaning a change to the way one thinks about a problem. The early AI research assumed that the problem would be solved by searching through a problem space. A problem space is defined by the states of a problem and the operators which move between the states. So the states can be thought of as nodes in a graph and the operators are the edges between them. This is called a state space search, where the initial state and the operators implicitly define the state space. To be complete, a problem representation must include the initial state, a goal description, the operators, and a set of predicates and objects from which the previous items are defined. To solve the problem a search is carried out though this space of states. The solution is a path in the graph from the initial state to a state that satisfies the goal description.

The AI type of problem reformulation moves from one state space problem description to another. This is called a "change of representation". The notion of changing from one problem representation of a state space to another problem representation of a state space, we will call the standard theory of problem reformulation, throughout the rest of this paper.

Not all state spaces are tractable, one might not be able to search through the space in a reasonable time. Of course, depending on how important the problem is or how quickly the solution is needed, "reasonable time" can grow and shrink. Some state spaces can be very large (e.g., the total number of chess positions is roughly 10^{120}). The purpose of reformulation is to move from one problem representation to another problem representation which is "more tractable", in that it takes less time to solve.

A "change of representation" (from one state space representation to another), alters the state and/or operator representation, to arrive at a new problem representation. Now you have two problem representations for the problem and you actually have three search spaces. There is the meta-

level search space that moves from one problem representation to another. Each of the individual problem representations has its own base-level search space in which to solve the problem.

This meta-level space lets a problem solving system move from one problem representation to another using a set of transformations. You could be given a set of hand-coded problem representations and just move between them or you could have generative transformations that create the new problem representations (e.g., the difference between searching an explicit graph or an implicit one). A meta-level search can be conducted through the space of problem representations, trying to find a more tractable representation or more specifically a more tractable representation for that particular problem solver.

The reason this meta-level search is required, is that there does not exist a single representation that is necessarily more tractable for all problem solvers. In previous research (Riddle, Holte, & Barley, 2011), we showed that even within PDDL domains, given the same problem solver, one problem representation was faster for one problem instance and another problem representation was faster for another problem instance. We showed the same trend between problem solvers where one problem solver preferred one problem representation for a particular problem instance while another problem solver preferred another problem representation. This motivates the need for a problem reformulation framework. If there was a perfect problem representation that would work with every problem solver and every problem instance, then once we found it we would never have to do problem reformulation again. We do not believe there is a perfect problem representation. There will always be the need to move from one problem representation to another problem representation.

3. Representing Problem Representations

In order to have a space of problem representations, we need a language in which we can represent them. We call this the representation language, and it had a critical influence on the early reformulation research. The notion of problem reformulation has been around for at least 40 years, but there have been very few systems over the years which have actually implemented these ideas. There was quite a bit of work in the 1980s in this area, some of it was only theoretical (Korf, 1980; Subramanian, 1989) and others that actually were implemented (Holte, 1988; Iba, 1989; Lowry, 1989; Nadel, 1990; Riddle, 1991). One of the major problems with this early work was that representation languages themselves were a new field in AI and there was not a consensus on what was necessary for a "good representation language". Different researchers were using different representation languages. This made it difficult to compare their results.

One way to create a research community with a shared agenda is to provide a common set of tools. In the 1980s, different researchers worked on problem reformulations but they each had their own representation language, and therefore the results were not comparable. PDDL (McDermott et al., 1998) is the planning domain definition language, which has been used by the domain independent planning community since the 1990s. A PDDL representation is defined by two files: a domain file and a problem instance file. The domain file contains the predicates and the actions. The problem instance file contains the objects, the initial state and the goal description. Examples of domain and problem instance descriptions are given later in this paper. We can now write problem reformulation transformations that change a problem representation represented in PDDL into

another problem representation represented in PDDL. This is one of the main reasons that it is now a good time for a resurgence in reformulation research.

PDDL is not a perfect language, but it is being carefully extended and made more powerful. It includes numeric fluents, durative/continuous actions, derived predicates, state trajectory constraints, preferences, and object fluents (Helmert, 2008). It is certainly not a powerful enough language for human-level reasoning. But the opportunity to do problem reformulation in a language with a strong community should not be missed.

The International Planning Competition (IPC) for domain independent planners is a biennial competition and has been run since 1998. This competition is based on the PDDL language. All the planners submitted become public domain software. This gives researchers in problem reformulation a publicly available set of domain independent problem solvers. Any domain independent planner can run on the generated PDDL representations. We can see which representation is better for which problem solver on which problem instance (Riddle, Holte, & Barley, 2011). There are a large number of different heuristics used in these competitions. We can explore which problem solver and which heuristics are favored for each representation of a domain or even individually for each problem instance.

In addition, there are new domains (each with 20 problem instances) created for every competition. This gives us a large number of different problem domains on which to test our problem reformulation transformations. Now, different researchers' problem reformulation operators can all be compared using the same space of domain independent PDDL planners. All transformations created by different research groups can be compared to one another. Reformulation researchers should take advantage of the PDDL language and publicly available domain-independent planners.

4. Transformations for Problem Reformulation

In order to define the meta-level space of problem representations, we need the nodes (problem representations) and the edges (transformation). In the last section we discussed the problem representations, in this section we will describe the transformations. The problem representations are not prespecified, but are automatically generated by the transformation operators. In Section 4.1, we will discuss the higher level single composite transformation that we have created, which guarantees that the new problem representation and the old problem representation both share certain properties.

There are four main types of component transformations. These transformations alter the objects, the predicates, the actions, or the initial state and/or goal description. We will not discuss the component transformations from which the composite transformations are created. These can be found in (Riddle, Barley, & Franco, 2013). In this paper we will focus on the composite transformation and how the meta-level search framework will work.

4.1 Composite Transformations

We are creating a meta-level search framework to search through the space of problem representations. The component transformations we discussed in (Riddle, Barley, & Franco, 2013), were at a very low level. Most of these transformations cannot be applied independently, because most of the

problem representations in that search space would be "broken". For example, in the gripper initial state discussed in Section 6, we transform the initial state representation from "(at ball4 rooma) (at ball3 rooma) (at ball2 rooma) (at ball1 rooma)" to "(count ball1 rooma 4)". If we change the initial state we also have to change the predicates and the operators otherwise the problem solver can no longer solve the problem,. If we just changed the initial state, this would not be a useable representation, no problem solver could search this space. You cannot alter a predicate without also altering all the actions that use that predicate!

If the meta-level search framework had to search through a space where 98% of the problem representations were unexecutable, then the framework would not be a viable proposition. Therefore we combine these low level component transformations into higher level composite transformations. The advantages of these higher level transformations is that at every point in the space, the problem representations are "executable" (i.e., "not broken"). They still might be better (or worse) than the previous problem representation, but they are all valid problem representations for this problem.

The composite transformations from one representation to another may preserve certain properties (like the same number of solutions) or they may not. While some of the transformations may always be desirable to apply, most of the transformations will only sometimes be desirable. This means that a search through a meta-level space of representations will be a critical component of the framework for applying these transformations.

Whenever possible these composite transformations maintain a one-to-one or a one-to-many mapping between a solution in the new problem representation and solutions in the old representation. This means that solutions found in the new representation can be transformed back into solutions in the original representation, if required.

4.2 Transforming the Reformulated Solution Back Into the Original Representation

In some cases, we are guaranteed that there is a one-to-many mapping of the solution produced in the new problem representation to a set of solutions in the original representation. For instance, in Section 6 we are guaranteed that corresponding solutions in both representations will have the same number of actions (the same solution length). In these cases, we are able to transform the solution backward by changing each action in the transformed representation's solution into a valid action in the original representation's solution. If there is a one-to-many mapping there is some search involved to find variable bindings that guarantee the solution is correct. This is typically much less search than would be needed to simply solve the problem in the original space. Namely the solution in the transformed representation gives a lot of structure to the search.

5. The Meta-level Search Framework

In the previous paper (Riddle, Barley, & Franco, 2013), we defined the meta-level search space. This consists of the representation of the states (i.e., problem representations) and the operators between them (i.e., transformation operators). Now we can describe the framework for searching over this meta-level space. Our framework uses generative transformations. These transformations generate new problem representations from the given problem representations. A meta-level search will traverse the space of representations to find a good representation for the current problem solver.

There are 3 main techniques we are exploring: 1) heuristic search, 2) adaptive problem solving, and 3) portfolio systems. We will discuss these in turn.

In Section 6 we will discuss one heuristic we can use in PDDL problems to predict when one representation will be better than the other. As we implement more composite transformations, we might be able to predict when one transformation will be better with a certain combinations of problem solvers and heuristics. I should caution that a lot more work remains. We have not explored every combination of problem solver and heuristic with these two representations. In addition we have only explored a single dimension of the space (number of objects versus number of locations). But there are many more dimensions (how many of the locations are used in the initial state, how many of the objects can be merged together by the transformation). All of these dimensions need to be explored. So much more experimentation is necessary to determine whether these heuristics are reliable.

Our second approach is to use an adaptive problem solver. RA* (Franco & Barley, 2008a; Franco & Barley, 2008b; Franco & Barley, 2009; Franco, Barley, & Riddle, 2013a; Franco, Barley, & Riddle, 2013b) uses sampling and a runtime model to predict which combination of heuristics is the "best" for solving one particular problem. It then uses that "best" heuristic combination to solve the rest of the problem. We can use the same technique for determining which of the two representations performs better. We now have a number of heuristic/representation combinations, and we can use sampling to determine which will perform better. Again much experimentation still remains. How good is this system at predicting the best combination? Can it do this and still exhibit savings over just running the old representation? Does this work equally well in all domains? Can this be extended to other transformations where there is no guarantee that the solutions have the same length?

The last option is a standard portfolio system. The meta-level framework could choose more than one representation and try to solve the problem, each of them in parallel (or by time slicing). This is a standard approach in the IPC, where several different problem solvers or one problem solver with several different heuristics are used in parallel. Again this system will certainly work with one or two transformations. But it will not scale up to a large number of alternative representations.

The search at the meta-level does add to the overall problem solving time. Currently we have a very small set of transformations, but as we develop more transformations, heuristics must be devised for determining which transformations to apply within this meta-level space.

This is a very powerful framework. In Section 6 we discuss a case study showing a transformation between two problem representations in the Gripper domain. The results of this study are very encouraging.

6. Gripper Domain Case Study

The case study transformation we explore in this section is a combination of merging objects, changing predicates and changing the initial state and goal description. We will look at a concrete example by examining the GRIPPER domain. Bear in mind that this transformation is actually implemented

```
(define (problem strips-gripper-x-1)
  (:domain gripper-strips)
  (:objects rooma roomb ball4 ball3 ball2 ball1 left right)
  (:init (room rooma) (room roomb) (ball ball4) (ball ball3)
        (ball ball2) (ball ball1) (at-roby rooma) (free left)
        (free right) (at ball4 rooma) (at ball3 rooma)
        (at ball2 rooma) (at ball1 rooma) (gripper left)
        (gripper right))
  (:goal (and (at ball4 roomb) (at ball3 roomb) (at ball2 roomb)
             (at ball1 roomb))))
```

Table 1. Gripper domain problem description in prob01.pddl

as a generic, domain-independent transformation. The transformed representations for the domain and problem instance and the transformed solution were all automatically generated.

The GRIPPER domain involves a robot "robby" with two grippers "left" and "right". It has X balls (the smallest problem shown in Figure 1 has 4 balls) which it must move from rooma to roomb. All the problems in this domain involve moving all the balls from rooma to roomb. They only differ in the number of balls. The complexity of the problem is increased by the fact that it can pick up each ball in either the left gripper or the right gripper. It does not matter which it uses in the overall solution, it does increase the complexity of the problem (especially for optimal planners).

Currently (with a single composite transformation) we have not set up the meta-level search framework. We have been analyzing which representation, the original or the transformed works better for a particular problem solver. Our current problem solver is A* with the LM-cut heuristic (Helmert & Domshlak, 2009) using the Fast Downward system (Helmert, 2006). This has been one of the top optimal planners in the IPC competition for the last few years. In the GRIPPER domain, no optimal planner performs well. Helmert states "If we apply two state-of-the-art optimal planning algorithms (Haslum, 2007; Helmert, Haslum, & Hoffmann, 2007) to the GRIPPER domain, neither of them can optimally solve more than 8 of the standard suite of 20 benchmarks within reasonable run-time and memory limits,..." (Helmert & Röger, 2008).

A* using the Fast Downward system has trouble with the original GRIPPER domain for two main reasons: it is an optimal planner and it is a grounded planner. Grounded planners find the GRIPPER domain difficult because there are roughly N^2 grounded pick operators when you have N balls and two grippers. In addition, optimal planners find it difficult to prove optimality in large search spaces.

In Figure 1 we give the timing results and nodes generated for the GRIPPER domain on the original and transformed representations. The problem solver cannot solve past problem prob07 in the original representation because it runs out of memory. The time includes the time of all transformation costs. The planner ran out of memory with ulimit set at -t 7200 -s 100000 -v 15000000.) So basically in the original representation the nodes generated and the time grows exponential with the solution length. In the transformed representation, the the nodes generated grow linearly with the solution length. While the timing appears to be constant, it actually grows quadratically.

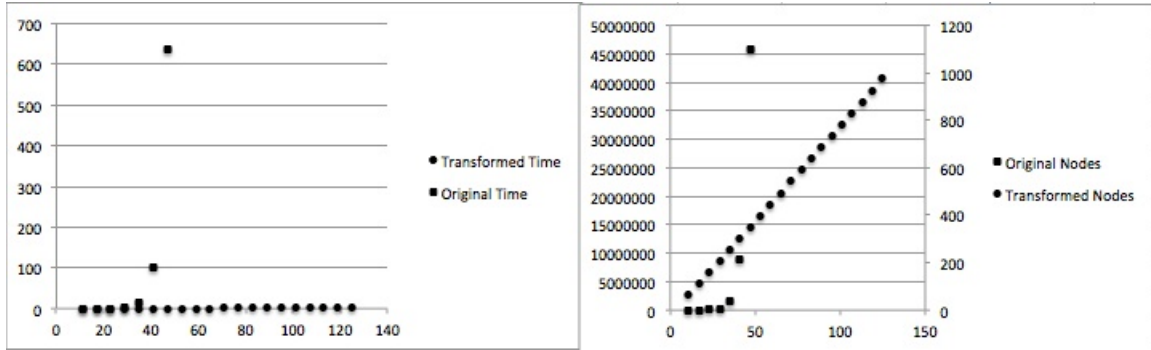


Figure 1. Time & Nodes for both Representations as function of Solution Length. In the right graph, the nodes in the original representation are plotted on the left axis, the nodes for the transformed representation are plotted on the right axis.

```
(define (domain gripper-strips)
  (:predicates (room ?r) (ball ?b) (gripper ?g) (at-robbly ?r)
               (at ?b ?r) (free ?g) (carry ?o ?g))

  (:action move
   :parameters (?from ?to)
   :precondition (and (room ?from) (room ?to) (at-robbly ?from))
   :effect (and (at-robbly ?to) (not (at-robbly ?from))))

  (:action pick
   :parameters (?obj ?room ?gripper)
   :precondition (and (ball ?obj) (room ?room) (gripper <?gripper)
                     (at ?obj ?room) (at-robbly ?room)
                     (free ?gripper))
   :effect (and (carry ?obj ?gripper) (not (at ?obj ?room))
                (not (free ?gripper))))

  (:action drop
   :parameters (?obj ?room ?gripper)
   :precondition (and (ball ?obj) (room ?room) (gripper ?gripper)
                     (carry ?obj ?gripper) (at-robbly ?room))
   :effect (and (at ?obj ?room) (free ?gripper)
                (not (carry ?obj ?gripper))))
```

Table 2. GRIPPER domain actions in domain.pddl

The original GRIPPER domain PDDL file is shown in Table 2, while the original problem instance file for the first problem is shown in Table 1. In this problem instance (see Table 1), there are four objects of type ball named **ball1**, **ball2**, **ball3**, and **ball4**. There are two objects of type room named **rooma** and **roomb**. There are two objects of type gripper named **left** and **right**. Because GRIPPER domain uses an early version of PDDL, the typing is done with unitary predicates. Later versions of PDDL have explicit typing. An initial state is given where robbly and all the balls are in


```

Transform:
  if NOT CanMerge
  then run old representation
  else
    change predicate in goal state
    change predicate in the initial state
    add predicates to initial state for
      empty locations (count X1 ?Y 0)
    add predicates to the initial state to
      handle arithmetic (more X+1, X)
    change predicate list in domain file
    change actions to refer to new predicate
    change parameters of action

```

```

CanMerge:
  objects X are not referred to in actions
    (as constant)
  objects X are all of the same type and
  objects X have the same predicates and
  additional values in the goal description

```

Table 3. Transforming PDDL Pseudocode

rooma and both grippers are empty. The goal description specifies that all the balls are in **roomb**. In the domain file (see Table 2), the things that don't vary from problem instance to problem instance are specified. The predicates are given, in this case they are the "type predicates" **room**, **ball**, and **gripper**; other unary predicates such as **at-robb**y and **free**; and relational predicates such as **at** and **carry**. The domain file also contains the action definitions. In the GRIPPER domain the actions are **move**, **pick**, and **drop**.

Our transformation system inputs the two PDDL files (the problem and the domain). The pseudocode is shown in Table 3. First it must determine whether there are any objects that can be merged. If it determines that there are no objects which can be merged, it will just run the planner on the original representation and return that solution. Two objects can be merged if:

- They are of the same type (using either PDDL typing or as in the old GRIPPER domain using singleton predicates).
- They appear in the same predicate in the goal description and have the same additional predicate values.
- Neither object is mentioned directly in any of the actions.

In the GRIPPER domain, this means that all the balls can be merged into 1 bag of "ball1"s at each location.

Now the problem is that most problem solvers cannot handle multiple objects with the same name and make sure they all get handled "correctly" in the goal state. Most problem solvers will put one ball1 into roomb and then stop, assuming the problem is solved.

```
( define ( domain gripper-strips )
  (:predicates ( room ?r ) ( ball ?b ) ( gripper ?g ) ( at-robbby ?r )
    ( count ?obj ?room ?num0 ) ( free ?g ) ( carry ?o ?g ) )

  ( :action move
    :parameters ( ?from ?to )
    :precondition ( and ( room ?from ) ( room ?to )
      ( at-robbby ?from ) )
    :effect ( and ( at-robbby ?to ) ( not ( at-robbby ?from ) ) ) )

  ( :action pick
    :parameters ( ?num2 ?num1 ?obj ?room ?gripper )
    :precondition ( and ( more ?num2 ?num1 ) ( count ?obj ?room ?num1 )
      ( ball ?obj ) ( room ?room )
      ( gripper ?gripper ) ( at-robbby ?room )
      ( free ?gripper ) )
    :effect ( and ( carry ?obj ?gripper )
      ( not ( count ?obj ?room ?num1 ) )
      ( count ?obj ?room ?num2 )
      ( not ( free ?gripper ) ) ) )

  ( :action drop
    :parameters ( ?num2 ?num1 ?obj ?room ?gripper )
    :precondition ( and ( more ?num1 ?num2 ) ( count ?obj ?room ?num1 )
      ( ball ?obj ) ( room ?room )
      ( gripper ?gripper ) ( carry ?obj ?gripper )
      ( at-robbby ?room ) )
    :effect ( and ( not ( count ?obj ?room ?num1 ) )
      ( count ?obj ?room ?num2 ) ( free ?gripper )
      ( not ( carry ?obj ?gripper ) ) ) ) )
```

Table 4. Transformed GRIPPER domain operators in domain.pddl

In problems like the GRIPPER domain, the balls are all going to have predicates in the goal description with identical values. Therefore these predicates all collapse into a single predicate instance and we have no way to distinguish them. Given this, it is more difficult to get the problem solver to exhibit the behavior we want. To achieve this, you can change the predicates so that instead of saying where each ball is, they count the number of balls at each location. This is a transformation that was explored by many of the problem reformulation researchers in the 1970s and 1980s. This allows you to easily make a goal which ensures all the "ball1"s are at their final location. This has the added benefit of making a problem representation with many fewer objects and a state space with far fewer states and edges.

Of course when you change the predicates, you must also change the actions, the initial state and the goal description to work with the new predicates. Looking back at Table 3, we must now change the **at** predicates (**at ball1 roomb**) (**at ball2 roomb**) (**at ball3 roomb**) (**at ball4 roomb**) to a single **count** predicate (**count ball1 roomb 4**) in the goal description. We must put a similar

predicate in the initial state (**count ball1 rooma 4**) as well as an additional predicate that states (**count ball1 roomb 0**). In addition we must add arithmetic predicates, such as (**more 0 1**) (**more 1 2**) (**more 2 3**) (**more 3 4**). These changes to the initial state and goal description can be seen in Table 5. Lastly we must change all the actions to refer to the new predicates instead of the old ones. For instance, the **pick** action has a precondition of (**count ?obj ?room ?num1**) which it removes in the effects and adds (**count ?obj ?room ?num2**) where (**more ?num2 ?num1**). These actions are shown in Table 4.

In problems where only some of the objects map to the same goal value, the composite transformation will make multiple bags. For instance, in the GRIPPER domain, if the goal description had two balls in **roomb** and two balls in **rooma**, then two balls would be retained each having two instances, giving a goal description of (**goal (and (count ball3 roomb 2) (count ball1 rooma 2))**).

The last thing our system does is transform the solution for the new problem representation back into a solution for the original problem representation. The solution, generated by a domain independent planner for the transformed representation, is shown on the left side of Table 6. The solution, automatically generated by our backward transformation to the original representation, is shown on the right side of Table 6. For this transformation there is a one-to-many mapping between the transformed solution and a set of original solutions. Moreover there is guaranteed to be a solution of the same length that is still optimal. To be more precise, given a sequence of actions that solves the problem in the transformed representation, we are guaranteed that it can be transformed into a solution in the original representation by going through a two step process. First, we remove those new action arguments which were introduced by the transformation. Second, we replace the occurrences of the new "merged object" in the actions' arguments by the appropriate original "mergee objects". There must exist a "mergee object" which can be placed as an argument in this action. For example, the solution shown on the left side of Table 6 was transformed into the solution on the right side by removing the first two arguments of the **pick** and the **drop** actions and by replacing the merged object, **ball4**, by **ball4** in the first action, by **ball3** in the second action, by **ball4** in the fourth action, by **ball3** in the fifth action, etc. To find the correct "mergee objects", a search must be performed. This is currently done with depth first search with backtracking, verifying each operator and generating the resulting state before moving on to the next operator. Although there may be many solutions in the original problem's representation which satisfy this mapping, our system finds a single set of bindings for the variables that makes a valid solution for the original representation. It returns the first solution it finds.

```
( define ( problem strips-gripper-x-1 )
  ( :domain gripper-strips )
  ( :objects 4 3 2 1 0 rooma roomb ball1 left right )
  ( :init ( room rooma ) ( room roomb ) ( ball ball1 )
    ( at-robby rooma ) ( free left ) ( free right )
    ( gripper left ) ( gripper right ) ( count ball1 rooma 4 )
    ( more 0 1 ) ( more 1 2 ) ( more 2 3 ) ( more 3 4 )
    ( count ball1 roomb 0 ) )
  ( :goal ( and ( count ball1 roomb 4 ) ) ) )
```

Table 5. Transformed GRIPPER domain problem description in prob01.pddl

(pick 3 4 ball4 rooma left)	(pick ball4 rooma left)
(pick 2 3 ball4 rooma right)	(pick ball3 rooma right)
(move rooma roomb)	(move rooma roomb)
(drop 1 0 ball4 roomb left)	(drop ball4 roomb left)
(drop 2 1 ball4 roomb right)	(drop ball3 roomb right)
(move roomb rooma)	(move roomb rooma)
(pick 1 2 ball4 rooma left)	(pick ball2 rooma left)
(pick 0 1 ball4 rooma right)	(pick ball1 rooma right)
(move rooma roomb)	(move rooma roomb)
(drop 3 2 ball4 roomb left)	(drop ball2 roomb left)
(drop 4 3 ball4 roomb right)	(drop ball1 roomb right)

Table 6. Solution in Transformed Space (left) and Original Space (right) for the GRIPPER domain

7. Summary Insights

The transformation currently works on 3 other PDDL domains: TRANSPORT-OPT11, NOMYSTERY-OPT11, and ELEVATORS-OPT11. In these domains, in some problems, there is no transformation done because no two objects are moving to the same location in the goal description. When a transformation is done, the total time taken in the transformed representation is always longer with the heuristic LM-cut in these 3 domains! So the question is what is different about the GRIPPER domain and these 3 other domains?

We have developed a transformation that creates a better representation any time there are substantially more objects than locations, and there are a large number of the objects are going to the same location. Even though the transformed representation always has a smaller blind search space size, there are problems where it makes the problem solver run slower. We have done two crossover experiments. In one experiment we took a GRIPPER problem and added more locations until the original representation solved the problem faster. In the other experiment we took a NOMYSTERY-OPT11 problem and added more packages until the transformed representation solved the problem faster. This shows that once the number of objects exceeds the number of locations by a "large enough" amount (and there are a large enough number of objects going to the same locations), the transformed representation does better. This leads us to believe a heuristic could be used to decide when to bother with transforming the problem representation.

Taking a broader view (more general than transportation problems), the transformed representation is good for domains with a lot of objects that can only have one value at a time (a ball or package can only be in one location at a time). Whereas the original representation is good for domains with a lot of objects that can have many values at once (a location can have many balls or packages at that location at once).

A number of interesting questions remain. What effect will these new PDDL representations have on different problem solvers? Can we alter the transformed representation so that it has the same state space reduction but does not cause the heuristics to be slower and less accurate? Can we find a "good" heuristic for the transformed representation?

In our experiments with GRIPPER and NOMYSTERY-OPT11 and TRANSPORT-OPT11 using LM-cut and iPDB, we have found that different heuristics get very different initial h-values on the

two representations. This tells us that the heuristics are much less accurate in some representations than in others. Can we find heuristics which inform us when a heuristic will do well on a particular representation?

In our previous work (Riddle, Holte, & Barley, 2011), we showed that the "best representation" for a problem varies from problem to problem using the same problem solver. In addition the "best representation" for a particular problem varies as you change from one problem solver to another. This tells us that the heuristics which predict what representation is likely to work best, must take into account the problem solver, the heuristic, and the particular problem to be solved.

8. Related Research

The early work on problem reformulation was theoretical and did not include implementations. (Amarel, 1961; Amarel, 1965; Amarel, 1968; Amarel, 1971; Newell & Simon, 1972; Korf, 1980). Several researchers developed automatic creation of macro-operators (Korf, 1985; Iba, 1989; Riddle, 1991). Lowry (1989), looked into problem reformulation as a type of automatic programming. Subramanian (1989) explored the importance of relevance in problem reformulation. Nadel (1990) explored 8 representations of the 4-queens problem. Preditis (1993) automatically generated admissible heuristics by automatically generating abstractions.

The most recent work on problem reformulation is by Fink (2003), who automates some changes of representation. The constraint propagation community has done a lot of work in the area of symmetry breaking (Walsh, 2012). In addition, research by Helmert (2009) focuses on turning PDDL into a concise grounded representation of SAS+. Further work on transforming problem representations has been done (Haslum, 2007; Helmert, 2006), they transform between PDDL and binary decision diagrams and causal graphs respectively.

9. Conclusions and Future Work

Now is an excellent time to return to the area of problem reformulation. A problem's representation can make it either easier or harder to solve a problem. We want to make an adaptive problem solver that can find a more tractable problem representation which it can then use to solve the problem. The transformations which provide the new problem representation should be generative.

We have a powerful framework that creates a flexible problem solver, by providing automatic transformations from intractable problem representations to easier problem representations. This allows many problem solvers to solve problems that they could not solve before, in reasonable time and space. We explored a case study of a composite transformation, presenting encouraging results in the GRIPPER domain where previously intractable problems can now be solved in a few seconds.

We have shown that you can affect a huge savings in problem solving time by spending a little time transforming the representation. This opens a brand new approach to improving problem solving systems. With the results varying so widely from one representation to another we think there will be increased interest in this area at ICAPS and in the IPC. Since the transformations produce PDDL files, they can be used seamlessly with any problem solving system. We plan to extend this work and develop a set of transformations and a method for searching through the space of transformations to decide which representation is the best choice for a given problem.

Our future research is currently going in two directions: a problem solver that uses multiple representations and developing more reformulations. Each of these will be addressed in turn. There are at least three obvious ways to use the representations we create. The first is to use them in portfolio type system where you just run each different representation and return the one that finished first. This will work fine as long as you only have a couple of representations, but once you get 4 or more representations this will probably not be a viable alternative. A second alternative is to use cross-over studies to predict when each will perform better, it is unclear how well this will work, but it is at least an option for exploration. The last possibility is to use the RA* system (Franco & Barley, 2008a; Franco & Barley, 2008b; Franco & Barley, 2009; Franco, Barley, & Riddle, 2013a; Franco, Barley, & Riddle, 2013b). This system uses sampling to determine the heuristic branching factor and the cost of applying heuristics, to determine which heuristic to use. The same technique can be used to determine which representation is more likely to solve the problem. This is especially true when the optimal solution path is the same length in both representations, it might be more difficult if the two representations produce different solution path lengths.

The second direction for future research is to create more transformations. The first approach is to do the same type of transformation in domains like SOKOBAN where the objects can never be at the same location at the same time. In these domains we do not need to make a bag of objects, we can just call all the objects S1, but we do have to take some care with the goal description in the transformed problem representation, because the predicate only takes one variable and so the objects can collapse. There are other standard reformulations we are looking at (like macro-operators). We will also look at extending the current transformation to work in cases where an object is spread over many predicates in the goal description, like in the WOODWORKING-OPT11 domain.

Acknowledgements

Thanks to Pat Langley for advice on earlier drafts of this paper.

References

- Amarel, S. (1961). An approach to automatic theory formation. *Principles of Self-Organization: Transactions. Pergamon Press, New York, ny.*
- Amarel, S. (1965). Problem-solving procedures for efficient syntactic analysis. *ACM Twentieth National Conference.*
- Amarel, S. (1968). On representations of problems of reasoning about actions. *Machine Intelligence 3.*
- Amarel, S. (1971). Representations and modeling in problems of program formation. *Machine Intelligence 6.*
- Fink, E. (2003). *Changes of problem representation: Theory and experiments.* Springer-Verlag.
- Franco, S., & Barley, M. (2008a). In situ reconfiguration of heuristic search on a problem instance basis. *2008 AAAI Workshop on Metareasoning: Thinking about Thinking.*
- Franco, S., & Barley, M. (2008b). Using sampling to dynamically reconfigure problem-solvers. *The First International Symposium on Search Techniques in Artificial Intelligence and Robotics.*

- Franco, S., & Barley, M. (2009). Predicting the optimal combination of pattern databases for solving a problem. *International Symposium on Combinatorial Search*.
- Franco, S., Barley, M., & Riddle, P. (2013a). In situ selection of heuristic subsets for randomization in ida* and a. *Heuristics and Search for Domain-Independent Planning* (p. 5).
- Franco, S., Barley, M., & Riddle, P. (2013b). A new efficient in situ sampling model for heuristic selection in optimal search. *Proceedings of the Australasian Joint Conference on Artificial Intelligence*.
- Haslum, P. (2007). Reducing accidental complexity in planning problems. *Proceedings of the International Joint Conferences on Artificial Intelligence* (pp. 1898–1903).
- Hayes, J., & Simon, H. (1974). Understanding written problem instructions. *Knowledge and Cognition*.
- Helmert, M. (2006). The fast downward planning system. *Journal of Artificial Intelligence Research*, 26, 191–246.
- Helmert, M. (2008). Changes in PDDL 3.1. Unpublished summary from the IPC-2008 website. <http://ipc.informatik.uni-freiburg.de/PddlExtension>. Retrieved November 2013.
- Helmert, M. (2009). Concise finite-domain representations for pddl planning tasks. *Artificial Intelligence*, 173, 503–535.
- Helmert, M., & Domshlak, C. (2009). Landmarks, critical paths and abstractions: What’s the difference anyway? *Proceedings of the International Conference on Automated Planning and Scheduling*.
- Helmert, M., Haslum, P., & Hoffmann, J. (2007). Flexible abstraction heuristics for optimal sequential planning. *Proceedings of the International Conference on Automated Planning and Scheduling* (pp. 176–183).
- Helmert, M., & Röger, G. (2008). How good is almost perfect?. *Proceedings of the Association for the Advancement of Artificial Intelligence* (pp. 944–949).
- Holte, R. (1988). *An analytical framework for learning systems*. Doctoral dissertation, University of Texas at Austin.
- Iba, G. (1989). A heuristic approach to the discovery of macro-operators. *Machine Learning*, 3, 285–317.
- Korf, R. (1980). Toward a model of representation changes. *Artificial Intelligence*, 14, 41–78.
- Korf, R. E. (1985). Macro-operators: A weak method for learning. *Artificial intelligence*, 26, 35–77.
- Lowry, M. R. (1989). *Algorithm synthesis through problem reformulation*. Doctoral dissertation, Stanford University.
- McDermott, D., Ghallab, M., Howe, A., Knoblock, C., Ram, A., Veloso, M., Weld, D., & Wilkins, D. (1998). Pddl-the planning domain definition language.
- Nadel, B. (1990). Representation selection for constraint satisfaction: A case study using n-queens. *IEEE Expert*, 5, 16–23.
- Newell, A. (1966). On the representations of problems. *Computer Science Research Reviews. Carnegie Institute of Technology, Pittsburgh, PA*.

- Newell, A., & Simon, H. A. (1972). *Human problem solving*, Vol. 14. Prentice-Hall Englewood Cliffs, NJ.
- Ohlsson, S. (1992). Information-processing explanations of insight and related phenomena. *Advances in the psychology of thinking*, 1–44.
- Pólya, G. (1957). *How to solve it: A new aspect of mathematical method*. Princeton University Press. Second edition.
- Prieditis, A. E. (1993). Machine discovery of effective admissible heuristics. *Machine learning*, 12, 117–141.
- Riddle, P. (1991). *Automatic Shifts of Problem Representation*. Doctoral dissertation, Rutgers University.
- Riddle, P., Barley, M., & Franco, S. (2013). Problem reformulation as meta-level search. *Proceedings of the Conference on Advances in Cognitive Systems*.
- Riddle, P. J., Holte, R. C., & Barley, M. W. (2011). Does representation matter in the planning competition? *Proceedings of the Symposium on Abstraction, Reformulation, and Approximation*.
- Simon, H. (1972). On reasoning about actions. In *Representation and meaning*. Englewood Cliffs, NJ: Prentice-Hall.
- Subramanian, D. (1989). *A theory of justified reformulations*. Doctoral dissertation, Stanford University.
- Walsh, T. (2012). Symmetry breaking constraints: Recent results. *Proceedings of the Association for the Advancement of Artificial Intelligence*.

How to Develop Fluid Intelligence via Metacognitive Self-Organization

Alexei V. Samsonovich

ASAMSONO@GMU.EDU

Krasnow Institute for Advanced Study, George Mason University, 4400 University Drive MS 2A1,
Fairfax, VA 22030-4444, USA

Abstract

Recent empirical studies provide evidence that human general fluid intelligence can be improved by certain training paradigms including games and puzzle solving that activate specific functional networks of the brain. This observation suggests that, similarly to the human brain, high fluid intelligence in artificial cognitive systems quite likely can be achieved by evolving and adaptively modifying specific functional components and mechanisms that give rise to high fluid intelligence during certain tasks. For this purpose, (a) the cognitive architecture should be made plastic and malleable, allowing for pruning and grafting of components and modification of their parameters driven by metacognitive evaluation of the local cognitive demands, and (b) a standard set of tests for fluid intelligence and training paradigms that are known to be able to improve fluid intelligence should be used. The expectation is that virtual agents possessing human-level fluid intelligence can be practically achieved using this approach.

1. Introduction

Among critical challenges for virtual agents on the road to their integration into the human society are creativity, intuition and human-level metacognitive abilities, such as the abilities to generate goals in novel situations and to produce believable behavior (Samsonovich, 2012). The term “fluid intelligence” refers to these and other intelligent capabilities that are independent of previously acquired specific knowledge or the background. More precisely, fluid intelligence G_f is regarded as the ability to understand and successfully deal with novel problems and situations, to identify patterns and relationships that can be extrapolated using logic, inductive reasoning and intuition, leading to a solution. Together with crystallized intelligence G_c that depends on the accumulated knowledge, G_f determines the general intelligence factor G (Cattell, 1987).

It was long believed that fluid intelligence couldn't be improved by training. On the contrary, recent empirical studies show exactly the opposite. One example is a highly cited empirical study concluding that working memory training using the dual N-back paradigm results in improvement of fluid intelligence test scores (Jaeggi et al., 2008). Many follow-up studies are devoted to this paradigm, e.g., <http://www.bostonglobe.com/lifestyle/health-wellness/2013/07/07/depressed-anxious-getting-older-there-app-for-all-that/oAflUdvS6pBL0hIs20mZFI/story.html>. Overall, empirical literature characterizing the relation between working memory capacity, its training, and fluid intelligence is quite inconsistent. The debate about the findings of Jaeggi et al. (2008) continues (e.g., Chooi & Thompson, 2012; Redick et al., 2012). E.g., Moody (2009) centers around the concern that the original findings reflect the effect of training on the speed of performance under pressure rather than on fluid intelligence. Another study shows, however, that

higher short-term memory span may not imply higher problem solving ability under pressure (Beilock & Carr, 2005), unless subjects talk aloud about their task (DeCaro, 2010). Many of these studies in question, however, are focused narrowly on the working memory capacity as the independent variable. The apparent inconsistencies can be explained when we acknowledge the fact that fluid intelligence depends on a distributed network of brain structures with diverse functionality, of which working memory is only one component. Therefore, success or failure to increase fluid intelligence in each particular paradigm depends on whether working memory is trained as a part of the network of brain areas supporting fluid reasoning (Preusse et al., 2011) or as an isolated component. This observation is the key to the idea of a new approach to intelligent agent development through self-organization of its architecture. At the same time, it should be understood that this idea is based on one possible explanation, which is not the only reasonable explanation.

The above consideration leads to the following question: despite the aforementioned controversy and in light of its suggested possible resolution, is it possible to use the same general principles of training in order to improve fluid intelligence in virtual agents? First of all, to be suitable for this approach, a cognitive architecture needs to be plastic and capable of self-modification, adaptation and self-organization driven by metacognitive evaluation of specific, localized cognitive demands. For example, high working memory load should result in an increase of working memory capacity. Frequent activation of specific cognitive mechanisms should result in an increased support for those mechanisms, and so on.

To support the above intuitive line of reasoning, here we provide a model in the form of a biologically inspired cognitive architecture (BICA) developed at George Mason University. This model explains the nature of the difference between the aforementioned conflicting results and predicts a substantial potential for improvement of the effect of training on fluid intelligence – in humans and, quite likely, in artifacts. This BICA model serves as the working hypothesis that can guide the metacognitively-driven self-organization of the system.

When mapped onto the brain, the BICA model provides likely neurophysiological correlates of fluid intelligence that define measures for our study. When implemented computationally, the BICA model will provide quantitative predictions for changes in the identified behavioral as well as physiological measures due to training.

2. BICA and Fluid Intelligence

A defining characteristic of fluid intelligence is that it applies to problems that are novel in their components, challenges, and objectives. In essence, each problem must be solved in a unique manner, using techniques tailored to the specific problem. This implies that fluid intelligence is mediated by multiple capabilities or mechanisms that are configured for each unique problem. Here we use the following terminology: fluid reasoning, understood as adaptive reasoning and problems solving (ARP), requires fluid intelligence (measured by standard tests as a cumulative characteristic G_f) plus other cognitive abilities, such as the ability to form complex verbal associations. Fluid intelligence in turn depends on a number of factors, including the size of working memory, the executive function, cognitive decoupling, and more.

According to the standard theory (Raichle et al. 2007; cf. Christoff, 2012), brain activity at rest (or activity independent of any task) is characterized by the default mode network (DMN). This network is suppressed during ARP, when the brain switches to the G_f network (GFN) supporting fluid intelligence. GFN involves many distributed parts of the brain and functional

components, and this is why activation of working memory (or even more narrowly, short-term memory) alone is not sufficient for enabling fluid intelligence through GFN activation: it may succeed or may fail, as the literature demonstrates, depending on the engagement of other vital functional components in the network. Therefore, it is necessary to activate the entire GFN in order to improve Gf. This is the main hypothesis underlying the proposed idea.

At a mathematical level, the above statements translate into the BICA model (Section 2.1) and its components (Figure 1) supporting fluid reasoning that map onto components of the Gf network in the brain. BICA is one of the models known as cognitive architectures (Gray 2007), many of which are biologically inspired. It was developed at George Mason University (Samsonovich & De Jong, 2005) and sometimes is referred to as GMU BICA. The mapping of BICA components into GFN provides the main prediction used in the intervention design and measures. We predict that activation of short-term memory alone is not sufficient for fluid intelligence. Instead, we need to engage certain functional components – elements of the BICA model, including components of working memory, semantic memory, the value system and the cognitive map.

The N-back paradigm (Kirchner, 1958) consists in detecting matches in a sequence of presented stimuli, comparing the current stimulus to the stimulus presented N steps back. Variations of the paradigm include fixed N and variable N that changes based on the subject performance. The process of solving the N-back paradigm by the BICA model is considered below (Figure 2). The diagram indicates that the working memory load (and therefore the required working memory size) will increase with the number N, and will increase even more as N will become variable. Moreover, the diagram shows activation of the value system in the case when incentive is used. The model also predicts activation of the spatial reasoning when the spatial element is present in the paradigm (not shown in Figure 2). All these components map to the brain as described below. Therefore, BICA model predicts activations of a distributed Gf network, and will guide the design of the intervention. More detailed predictions should become available based on the proposed pilot study.

2.1 Overview of the model

The general biologically inspired cognitive architecture (BICA) model is described in Figure 1. This model substantially extends the extended Baddeley working memory model (Baddeley et al., 2009). Its components have been mapped to brain areas (Samsonovich & De Jong, 2005; Samsonovich, De Jong & Kitsantas, 2009). The BICA model is based on three main building blocks: (i) a schema, which is a universal element of all representations in this model; (ii) a mental state, which is populated by bound instances of schemas and represents the content of immediate awareness of a certain subject in a certain mental perspective; and (iii) a cognitive map, which uses an abstract semantic space to represent relations among mental states, schemas and their instances.

Components of the BICA model (Figure 1) include working, semantic, episodic, procedural, and sensory memories, plus a value system and a cognitive map. All details of the BICA model cannot be discussed here. The core framework can be summarized by the following set of self-explanatory tuples, some elements of which can be empty (Samsonovich, 2013):

- semantic map = (semantic space, mental states and schemas, their allocations)
- mental state = (attributes, schemas)

- schema = (attributes, terminal nodes, internal nodes, links)
- node = (attributes, reference to schema)
- attributes = (category, perspective, attitude, appraisal, bindings, activation, ...)

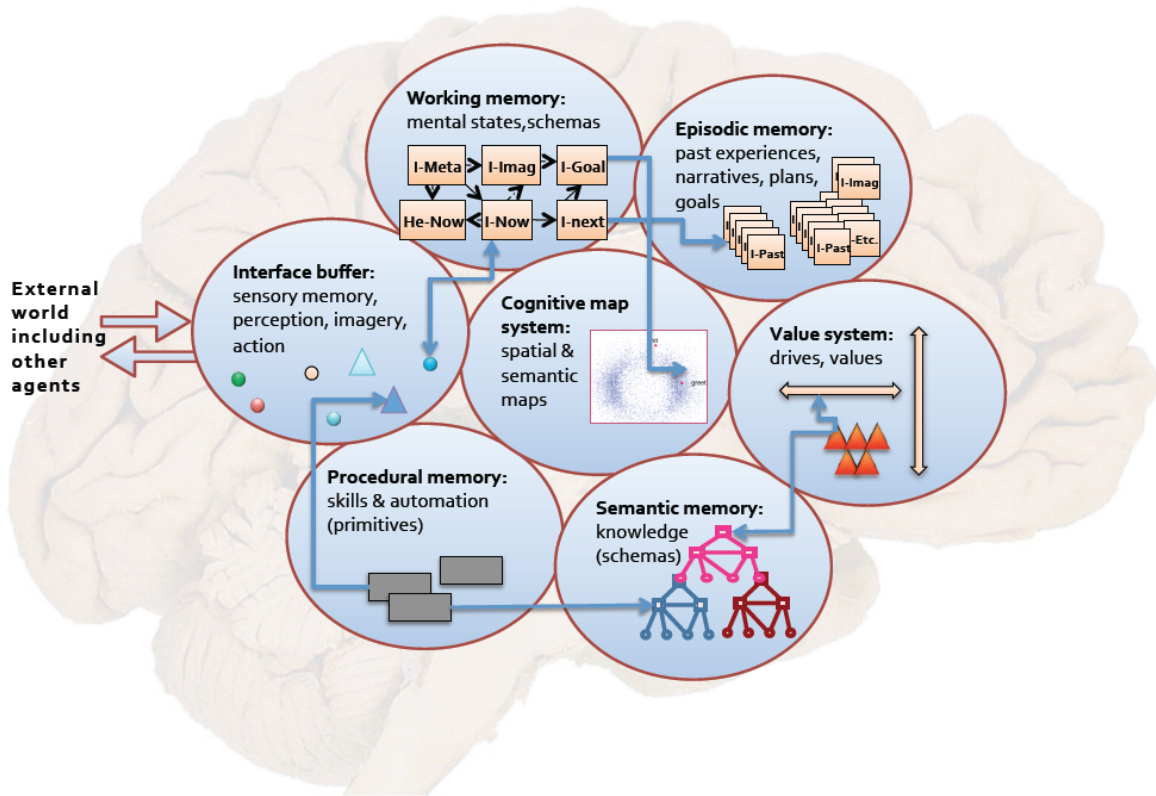


Figure 1. General view of the BICA model as 7 interconnected components (from Samsonovich, 2013).

2.2 Metacognitive component

The metacognitive component is not shown explicitly in Figure 1, because it is distributed through other components of the architecture. In general, this component will provide the cognitive system with the ability to detect and handle anomalies, changes, or opportunities. It will implement both monitoring of cognitive activities in BICA and control of these activities. In addition, it will implement cognition about such activities and learning through adaptive restructuring of the architecture.

In contrast with the majority of cognitive architectures, in BICA, the metacognitive component can be naturally grounded in basic principles and elements of the architecture. The

main basis for it is the structure of working memory in BICA, which is partitioned into mental states. As mentioned above, a mental state represents a snapshot of awareness of the agent, and can be viewed as a model of an agent on its own. Therefore, multiple mental states that are co-active in working memory can monitor and operate on each other, instantiating various forms of metacognition. In BICA, a metacognitive mental state that monitors and controls other active mental states in working memory is called I-Meta. It creates a token for each monitored mental state, and can be used, e.g., to optimize interactions among mental states.

In our case, the task for I-Meta is to implement activity-driven self-organization. This task requires not only monitoring of mental states themselves, but also awareness of their substrates – parts of the system that need to be expanded or further augmented with available resources in response to detected demands. Details of operation of the metacognitive component should be developed based on empirical studies.

2.3 Mapping BICA to the brain

Components of the BICA model can be roughly mapped onto the brain as follows.

- *Working memory*: activity in ventrolateral prefrontal cortex (VLPFC, BA 45), dorsolateral prefrontal cortex (DLPFC, BA 46), dorsal parietal (BA 9, 46), dorsal frontal (BA 10)
- *Working memory*, figural reasoning: dorsal parietal cortex (BA 7), ventral frontal parietal cortex (BA 40).
- *Cognitive map* for episodic and spatial reference memory: hippocampus, parahippocampal gyrus (entorhinal, perirhinal and retrosplenial cortices), lateral parietal cortex, nuclei of diencephalon
- *Cognitive map* for utility matrix in decision making: lateral intraparietal cortex (LIP)
- *Value system*, absolute value: orbitofrontal cortex; relative reward value: striatum
- *Value system*, emotional network: anterior cingulate cortex, amygdala, orbitofrontal cortex, hypothalamus, insula,
- *Episodic memory*: synaptic weights in/between the hippocampus and extrastriate neocortices
- *Semantic memory*: medial temporal lobe, parahippocampal, prefrontal, parietal cortices
- *Interface buffer*, output and imagery: premotor and motor cortices, cerebellum;
- *Interface buffer*, input and imagery: primary sensory cortices and related structures of their thalamocortical loops
- *Procedural memory*: specialized neocortical areas, including visual, auditory, language (Broca, Wernicke), motor and premotor cortices and related structures of their thalamocortical loops, plus the cerebellum

2.4 Dynamics and general predictions

At the core of BICA dynamics is the standard cognitive cycle including perception, cognition and action. Schemas invoked by sensory input and working memory content populate mental states in working memory and bind to other schemas, resulting in new elements of awareness, intentions and initiation of actions. This picture, as outlined below, captures various aspects of fluid reasoning and generates expectations for physiological measures.

An expectation based on previous studies and the BICA model is that during a challenging cognitive task within the intervention, DMN will be suppressed due to activation of a fluid intelligence network, GFN. As a result, a new network of brain activity GFN will be stabilized

over the training period, resulting in an increase of fluid intelligence Gf. The new network may differ by its elements (i.e., topologically) and/or by the distribution of weights of connections. Our intervention will, in various combinations, exercise/train the subnetworks involved in realizing selected mechanisms required for Gf. In order to ensure integration of the necessary cognitive functionality in the network, we include additional elements in the game design, as described below.

When the BICA model is solving the N-back task, one active mental state in working memory is used to maintain awareness of each presented item and its expected matching N steps forward (using the schema of an N-back match). In one plausible scenario (Figure 2), the number of mental states used for solving the task that need to be co-active is N+1, the total number of mental states in working memory will be slightly higher due to the fact that mental states are not deactivated instantly, plus there may be other mental states. The total number of mental states in working memory will inevitably fluctuate (details will become clear upon implementation and simulation of the BICA model in this paradigm). When N is increased by one, the number of mental states will increase accordingly: it will be necessary to keep one more mental state in working memory to solve the task. Assuming that each mental state has its own independent neural substrate associated with brain activation, we may expect that, for a typical value of N (e.g., 1 to 3), each step to a higher N will be associated with an increase in working memory size by 15%-30%. With a switch to a variable N paradigm with an incentive to reach higher N, the challenge is higher, because the subject should hold many more mental states and schemas in mind to get ready for a possible high-N match. In this case, based on similar considerations, the working memory expansion is expected to go higher - to 30% - 50% and more. Training of this sort will have its effect on working memory span, and will translate into an increase in VLPFC differential BOLD signal during subject's engagement in another fluid intelligence task – the Raven's APM, done in the scanner. Fluctuations of the number of mental states cannot be estimated precisely before implementation and simulation of the BICA model in these paradigms. This logic, however, predicts a reasonable effect size of training on the neurophysiological measure (which is the difference between GFN and DMN activities) and on the behavioral score in APM. The latter conclusion is further supported by the finding of Jaeggi et al. (2008) of a significant increase of Gf scores due to N-back training and by related findings.

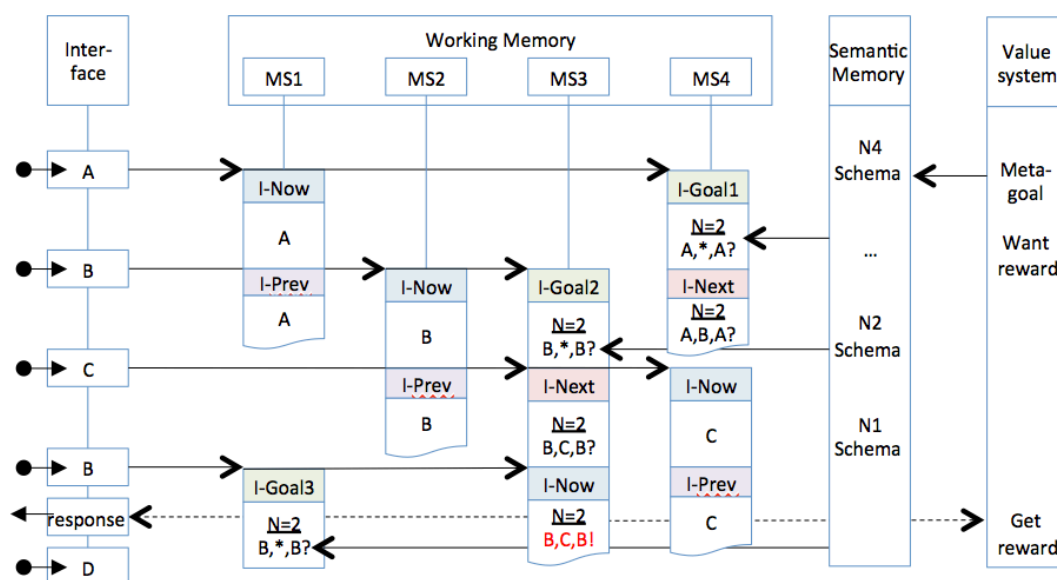


Figure 2. A simplified UML sequence diagram showing one possible strategy of solving N-back task for $N=2$ in the BICA model. A,B,... (left column) represent sequential stimuli. The goal is to match N back, currently with $N=2$ (the N2 schema is activated by incentive). The meta-goal for which the subject is rewarded is to increase N. At the meta-level it is a “spatial navigation task” described in (Samsonovich & Ascoli 2005), not represented here. With a fixed $N=2$, having 4 co-active mental states in working memory is sufficient for achieving the goal, as the diagram shows. With a variable N, more mental states would be required to achieve the goal and to make progress toward meta-goal (not shown here).

Selection of schemas is guided by their fitness into the working scenario, which is understood as the main sequence of mental states extending into the future – to the goal mental state, when incentive is present. Therefore, the introduction of incentive should activate the working scenario and therefore increase and organize activity of working memory in BICA. The goal state is created by activation of the target schema (e.g., completion of a level in a game), placing it in a mental state, and giving this mental state the status of a goal. Then the rest of cognition and behavior of the subject can be viewed as a process of navigation toward the goal on a cognitive map, accompanied by exploratory activations of the map that reveal utility values (Ascoli & Samsonovich, 2013). Therefore, we expect that incentive will activate corresponding parts of the brain (e.g. LIP: see the mapping above). Precise quantitative estimates will be made upon implementation and simulation of the BICA model tailored for the selected game paradigm.

In the N-back paradigm, the subject needs to hold multiple mental states in working memory, temporarily storing candidate future matches for the specified value of N. When the value of N varies and is not known a priori to the subject before each step, the challenge increases: even more schemas and mental states should be kept active in working memory. Therefore, these

conditions are expected to increase working memory activation (VLPFC, DLPFC, dorsal parietal areas – see above) and improve working memory span in general.

2.5 Strategy of demand-driven self-organization of BICA

The computationally implemented BICA model, when solving a set of standard tests for fluid intelligence, will reveal specific cognitive mechanisms and virtual networks involved in the process of solution, as the above example shows (Figure 2). In order to use the strategy of demand-driven modification and self-organization of the architecture, a metacognitive component is required to detect the activated patterns and to implement modifications of the architecture on the fly.

The analogy with the human brain and human behavioral and neuroimaging data suggests that by reinforcing and augmenting those specific activities, involved components and their interactions one should be able to increase general fluid-intelligent capabilities of the architecture. The same result may be difficult to achieve using deductive logic and traditional engineering.

Human data suggest that improvement achieved in one paradigm may be transferred to other domains and paradigms as well. For example, when the BICA model is used to solve Raven's Advanced Progressive Matrices (APM), activities and changes resemble those that occur during solving a variable-N-back task, etc. Indeed, the main cognitive process in most cases consists in applying various schemas to the perceived image, which predicts activation of semantic memory, in addition to the activation of working memory. Episodic memory is also used: subjects learn from their experience how to solve matrices. In addition, reasoning about matrices or sequences involves spatial cognition, figural reasoning, imagery, and the value system (motivation), thus engaging the corresponding brain areas (section 2.2) or architectural components.

These general predictions of the BICA model are common for many paradigms related to fluid intelligence. Indeed, when applied to a variable-n-back game paradigm (based on Jaeggi et al., 2008), the BICA model predicts patterns of brain activation during execution of the task including working, semantic and episodic memory components. Arguably, the BICA model can in general account for virtually all the key mechanisms of human higher cognition that support fluid intelligence. Examples include the following:

- Inference of the elements of a problem. This may be passive and achievable by simple pattern recognition or it may involve active perturbations to the problem context and statistical bootstrapping.
- Inference of the properties/affordances of the elements and their relationships to each other and to the agent. Again, this may be simple and direct or it may require interaction, temporal inferences, etc.
- Identification of the objective in terms of the elements of the problem and sub-objectives, such as the satisfaction of constraints and avoidance of obstacles and threats.
- Identification of solutions to the objective and sub-objectives. This may involve deduction, inference, planning, anticipating outcomes of interactions and causation.
- Working memory mechanisms that maintain the cognitive state and the ability to reason over the elements of the problem space.
- Episodic memory mechanisms that for a given situation identify relevant outcomes of past actions providing validation or support of reasoning.
- Simultaneous, multi-objective satisfaction and/or threat avoidance.

3. Behavioral Assessment of Fluid Intelligence

One of the key questions related to this approach is how to monitor changes of fluid intelligence in the system. Arguably, standard tests and techniques can be used for this purpose. There has been a fair amount of discussion about pros/cons of various measures of fluid intelligence. Given below are examples of available standard measures of fluid intelligence that can be used to assess the results of training – or during the training.

Raven’s Advanced Progressive Matrices (APM) Set I and II is published as several editions (Raven 1990; Raven, Raven & Court, 1998). Including an example of the actual test here is not possible due to the copyright restrictions. An example of a test that looks similar to APM can be retrieved from <http://forums.xkcd.com/viewtopic.php?f=3&t=41645>. Another similar example is shown in Figure 3. The figure also shows how the BICA architecture may solve this test by activating relevant schemas corresponding to patterns of alternating direction and progressive rotation.

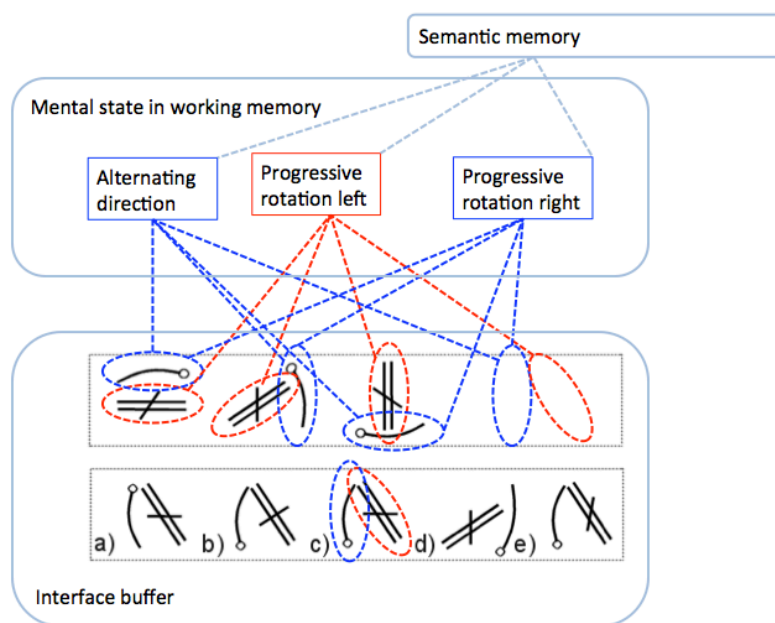


Figure 3. The BICA model solving a test for fluid intelligence that is similar to APM.

Listed below are several other examples of tests for fluid intelligence. all of them can be solved using BICA and can be used for training BICA. Wechsler’s Adult Intelligence Scale includes WAIS-IV and WASI tests that are available from <http://www.pearsonassessments.com/HAIWEB/Cultures/en-us/Productdetail.htm?Pid=015-8980-808>. Including an example of the actual test here is not possible due to the copyright restrictions. The Wisconsin Card Sorting Test (WCST) is a widely used, standard psychological test of "set-

shifting", i.e. the ability to display flexibility in the face of changing schedules of reinforcement (Berg, 1948; Monchi et al., 2001). Also, Bochumer Matrizen-test (BOMAT) is a standard test for fluid intelligence available from <http://www.testzentrale.de>.

Concluding Remarks

Development of a computational understanding of complex social phenomena, like emergence of affective relationships in small groups (possibly including virtual agents) and social decision making, is vital for successful control of unexpected situations, team management, policy making, and many other practical needs. This challenge requires a human-level cognitive architecture (intelligent agent model) possessing fluid intelligence, while the state of the art in artificial intelligence represented by popular cognitive architectures on average remains below the desired level.

The present work proposed an approach to solving the challenge based on activity-driven self-organization of the architecture, inspired by cognitive training of the human brain. The model suitable for self-organization can be constructed based on the BICA model, and, in particular, its new version – the eBICA model (Samsonovich, 2013) that captures the main principles and functional organization of the human brain-mind underlying complex social and emotional human behavior. It enables modeling of complex social phenomena based on first principles.

Indeed, recent empirical studies provide evidence that human general fluid intelligence can be improved by certain training paradigms including games and puzzle solving that activate specific functional networks of the brain. This observation suggests that, similarly to the human brain, high fluid intelligence in artificial cognitive systems quite likely can be achieved by engineering, evolving and adaptively modifying the same functional components and mechanisms that give rise to high fluid intelligence in vivo, when combined into the specific network. For this purpose,

- (a) the cognitive architecture should be made plastic and malleable, allowing for pruning and grafting of components and modification of their parameters driven by metacognitive evaluation of the local cognitive demands, and
- (b) a standard set of tests for fluid intelligence and training paradigms that are known to be able to improve fluid intelligence should be used.

The expectation is that virtual agents possessing a human level of fluid intelligence can be practically achieved based on this approach, with training and evaluation paradigms including games, meta-games, problem solving, team collaboration and competition, and more. Specific paradigms, tasks and goals may include examples like: using virtual agents to reproduce typical human social behavior in given settings; achieving a human level in interactions and emergent social relationships among virtual agents and human participants in various settings (the criterion could be passing a Turing test); efficiently controlling human behavior in a group of agents using interventions based on predictive BICA modeling. Expected impacts include theoretical, technological and social advances.

Acknowledgements

Some of the ideas of this work were developed in collaboration with Drs. Kevin McCabe, Kenneth De Jong, Dennis McBride, Scott Martin, Kathryn B. Laskey, Charles Peck, Anastasia Kitsantas, Kenneth C. Olson, and others.

References

- Ascoli, G. A. and Samsonovich, A. V. (2013). A spiking-network cognitive architecture inspired by the hippocampus. *Biologically Inspired Cognitive Architectures*, 3: 13-26. DOI: <http://dx.doi.org/10.1016/j.bica.2012.11.002>.
- Baddeley AD, Eysenck M, Anderson MC (2009). *Memory*. New York: Psychology Press.
- Beilock, S.L., and Carr, T.H. (2005). When high-powered people fail: Working memory and “choking under pressure” in math. *Psychological Science* 16:101-105.
- Berg, E.A. (1948). A simple objective technique for measuring flexibility in thinking *Journal of General Psychology* 39: 15-22.
- Cattell, R. B. (1987). *Intelligence: Its structure, growth, and action*. New York: Elsevier Science.
- Chooi, W., and Thompson, L. (2011). Working memory training does not improve intelligence in healthy young adults. *Intelligence* 40, 531–542.
- Christoff, K. (2012). Undirected thought: Neural determinants and correlates. *Brain Research*, 1428 (SI): 51-59.
- DeCaro, M.S., Rotar, K.E., Kendra, M.S., & Beilock, S.L. (2010). Diagnosing and alleviating the impact of performance pressure on mathematical problem solving. *Quarterly Journal of Experimental Psychology*, 63(8):1619-1630.
- Gray, W. D. (Ed.) (2007). *Integrated Models of Cognitive Systems. Series on Cognitive Models and Architectures*. Oxford, UK: Oxford University Press.
- Jaeggi, S.M., Buschkuhl, M., Jonides, J., Perrig, W.J. (2008) Improving fluid intelligence with training on working memory. *Proc Natl Acad Sci USA*. 105(19), 6829-6833.
- Jaeggi, S.M., Studer-Luethi, B., Buschkuhl, M., Su, Y.-F., Jonides, J., Perrig, W.J., 2010. The relationship between n-back performance and matrix reasoning—implications for training and transfer. *Intelligence* 38, 625–635.
- Kirchner, W. K. (1958). Age differences in short-term retention of rapidly changing information. *Journal of Experimental Psychology*, 55 (4): 352-358.
- Monchi, O., Petrides, M. Petre, V., Worsley, K., & Dagher, A. (2001). Wisconsin card sorting revisited: Distinct neural circuits participating in different stages of the task identified by event-related functional magnetic resonance imaging. *The Journal of Neuroscience*, 21(19), 7733-7741.
- Moody, D.E. (2009). Can intelligence be increased by training on a task of working memory? *Intelligence*, 37: 327-328.
- Preusse, F., van der Meer, E., Deshpande, G., Krueger, F., and Wartenburger, I. (2011). Fluid intelligence allows flexible recruitment of the parieto-frontal network in analogical reasoning. *Frontiers in Human Neuroscience* 5: 22, doi: 10.3389/fnhum.2011.00022.
- Raichle, M.E. and Snyder, A.Z. (2007) A default mode of brain function: a brief history of an evolving idea. *Neuroimage* 37(4), 1083–1090; discussion 1097–1089.
- Raven, J. C. (1990). *Advanced progressive matrices: Sets I, II*. Oxford: OxfordUniv Press.
- Raven, J., Raven, J. C., and Court, J. H. (1998). *Manual for Raven’s Progressive Matrices and Vocabulary Scales: 1998 Edition. Introducing Parallel Versions of the CPM and SPM together with a More Powerful Version of the SPM (SPM Plus)*. San Antonio, Texas: NCS Pearson, Inc.

- Redick, T.S., Shipstead, Z., Harrison, T.L., Hicks, K.L., Fried, D.E., Hambrick, D.Z., Kane, M.J., & Engle, R.W. (2012). No evidence of intelligence improvement after working memory training: A randomized, placebo-controlled study. *Journal of Experimental Psychology: General*. Advance online publication, doi: 10.1037/a0029082.
- Samsonovich, A. V. (2012). On a roadmap for the BICA Challenge. *Biologically Inspired Cognitive Architectures* 1: 100-107.
- Samsonovich, A. V. (2013). Emotional biologically inspired cognitive architecture. *Biologically Inspired Cognitive Architectures*, 6: 109-125
- Samsonovich, A. V. and Ascoli, G. A. (2005). A simple neural network model of the hippocampus suggesting its pathfinding role in episodic memory retrieval. *Learning & Memory* 12 (2): 193–208.
- Samsonovich, A. V. and De Jong, K. A. (2005). Designing a self-aware neuromorphic hybrid. In K.R. Thorisson, H. Vilhjalmsson, and S. Marsela (Eds.). *AAAI-05 Workshop on Modular Construction of Human-Like Intelligence: AAI Technical Report*, volume WS-05-08, pp. 71–78. Menlo Park, CA: AAAI Press.
- Samsonovich, A. V., De Jong, K. A., and Kitsantas, A. (2009). The mental state formalism of GMU-BICA. *International Journal of Machine Consciousness* 1 (1): 111-130.