An Architecture for Adaptive Algorithmic Hybrids

Nicholas Cassimatis[1], Magdalena Bugajska[1], Scott Dugas[1], Arthi Murugesan[1], Paul Bello[2]

[1]Department of Cognitive Science
Rensselaer Polytechnic Institute
Troy, NY 12180
{cassin, bugajm, dugass, muruga}@rpi.edu

[2]Air Force Research Laboratory
Information Directorate
Rome, NY 13441
Paul.Bello@rl.af.mil

Abstract

We describe a cognitive architecture for creating more robust intelligent systems by executing hybrids of algorithms based on different computational formalisms. The architecture is motivated by the belief that (1) most existing computational methods often exhibit some of the characteristics desired of intelligent systems at the cost of other desired characteristics and (2) a system exhibiting robust intelligence can be designed by implementing hybrids of these computational methods. The main obstacle to this approach is that the various relevant computational methods are based on data structures and algorithms that are very difficult to integrate into one system. We describe a new method of executing hybrids of algorithms using the *focus of attention* of multiple modules. This approach has been embodied in the Polyscheme cognitive architecture. Systems based on Polyscheme can integrate reactive robotic controllers, logical and probabilistic inference algorithms, frame-based formalisms and sensor-processing algorithms into one system. Existing applications involve human-robot interaction, heterogeneous information retrieval and natural language understanding. Systems built using Polyscheme demonstrate that algorithmic hybrids implemented using a focus of attention can (1) exhibit more characteristics of intelligence than individual computational methods alone and (2) deal with problems that have formerly been beyond the reach of synthetic computational intelligence.

Introduction

We describe a cognitive architecture for creating more robust intelligent systems by executing hybrids of algorithms based on different computational formalisms.

There are several properties that we desire of intelligent systems. Each of these properties is exhibited by some algorithm, but often at the cost of one of the other properties. For example, many search algorithms and many probabilistic inference algorithms for graphical models are general insofar as a wide variety of problems can be reformulated so that these algorithms can solve them. They are flexible in that when small changes are made to the knowledge bases or models these methods use, they correctly update their inferences. However, for larger problems, these algorithms become prohibitive in time and/or space. They thus trade efficiency for scalability. Reactive and behavior-based systems trade generality for speed and dynamism. These systems can quickly adapt their behavior to new information about the environment, but are generally not capable of making many kinds of complex inferences and plans that many reasoning or planning algorithms can. Approaches that are based on more complex and structured knowledge representation schemes such as frames (Minsky, 1975), scripts (Schank & Abelson, 1977) and cases (Kolodner, 1993) make the same tradeoff. They can make inferences and plans that would take more general algorithms too much time. However, frames, scripts and cases often do not work in situations that are even slightly different from those for which they were created.

Because of such tradeoffs, it has been difficult to create a single system that exhibits all the desired characteristics of intelligent systems. One approach to this problem is to create systems based on hybrids of these algorithms. This paper presents a cognitive architecture, called Polyscheme, for implementing and executing such hybrids. Many methods of integration involve loosely-coupled collections of modules or hybrids of only a small fixed set of algorithms. Although Polyscheme includes modules that can encapsulate algorithms, it also enables algorithms to be implemented through sequences of "attention fixations" (each involving all the modules) called "focus traces". By interleaving these focus traces, Polyscheme can execute hybrids of algorithms where each step of each algorithm can potentially be assisted by the others. As explained below, this form of integration enables systems that have previously not been feasible.

Unifying principles

We can motivate an architecture for integrating hybrids of multiple classes of algorithms by recognizing that even though they are based on very different computational formalisms, they share many common elements.

The following are some formal preliminaries. Strings of the form *P(... x_i ..., t,w)* are called propositions. They say that relation *P* holds over arguments x_i during temporal interval *t* in world *w*. A world is a history (past, present and future) of states. *P/w* refers to a proposition like *P* except for having *w* is its world argument. *E* ("eternity") is the temporal interval such that all other temporal intervals occur during it. *R* is the real or actual world. Terms can refer to the same object. This is indicated with *Same(x,y,E,w)*. *T*, *F and U* ("unknown") represent true and false truth values for a proposition.

Propositions are used only to characterize certain aspects of Polyscheme and as an interlingua between Polyscheme modules. Polyscheme is not a "logical" system insofar as it can use non-logical data structures and its computation is not predominantly deductive or confined to manipulating formulae in some logical language.

Common function principle

It is possible to characterize algorithms from very different formal frameworks as executing sequences of common functions. In this case "function" refers not to a mathematical function, but to the purpose of an algorithm as in "the function of a sorting algorithm is to order the elements of a collection". These functions are described with along with notation we will use to refer to them.

Store information. Given information about P's truth value, store it. `Store(P,TV)`.

Offer opinion. Return a truth value for a proposition. `OpinionOn(P,TV)`.

Forward inference. Given some knowledge, infer what follows. `ForwardInference(BK)`, where `BK = { ... (`P_i`, TV`$_i$`) ... }`, returns a list of proposition paired with their respective truth values.

Request information/Subgoal. In order to know about something, take actions to learn about it. Given `P`, return a set of propositions such that information about their truth will help infer `P`'s truth. `RequestInformation(P)`.

Identity. For any object, find other objects that it might be identical to. Where `O` refers to an object, `BK` is as above and `W` is an alternate world, return a set of objects that might be identical to `O` in `W`. `Matches( O, BK, W)`

Represent alternate worlds. The ability to represent and make inferences about alternate states of the world is implicit in all of the above common functions since they all involve propositions with worlds as arguments.

The following are examples of how some important algorithms can be implemented using common functions.

A variant of Markov Chain Monte Carlo simulation (operating over a Bayesian network with Boolean state variables) can be implemented thus, where `ForwardInference(BK)` samples a value for P given the value of variables in its Markov blanket in `BK`.

Represent a state variable as a proposition. Let P_i be the state variables.
Start with an assignment of truth values to those propositions { ... P_i/w_0 ...}.
For i = 1 to MAX-SIMULATIONS:
  For each `j`:
    `Store(Pj, ForwardInference({`
    `(P1/wj-1, OpinionOn(P1/wi-1)) ...`
    `(Pj-1/wi-1,OpinionOn(Pj-1/wi-1) ))`
`Prob( P )` = the proportion of worlds, `w`, in which `P/w` is true.

WalkSAT (Selman, Levesque, & Mitchell, 1992) search to satisfy a CNF constraint `C` can be represented in the following manner, where `ForwardInference(BK)` returns a list of one ordered pair `(P,TV)` where `P` is the proposition such that flipping its truth value will make the most clauses in `C` true.

For each proposition, `P`, `Store(P, Random(T,F))`
For I = 1 to MAX-FLIPS
If `OpinionOn(C) = T`
  Return { ... `(Pi, OpinionOn(Pi)` ... }
Otherwise
  With probability *p*,
    `Store(R, not(OpinionOn(R)))`
  Otherwise
    `Store(First ForwardInfrence(BK))`,
    where `BK` is the set of ordered pairs of each proposition with its truth value.

Case-based reasoning can be added to WalkSAT by modifying `ForwardInference` above to return propositions that were true in situations whose similarity to the current situation exceeds some threshold.

These examples illustrate how algorithms developed within very different formal frameworks can be characterized as sequences of common function executions.

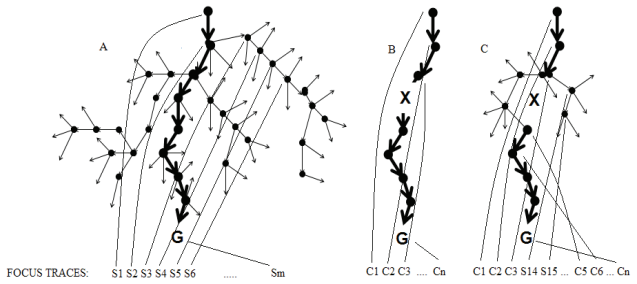

Figure 1. Algorithms and their focus traces. The focus trace (C) for the hybrid of case-based reasoning and search is a combination of the focus traces for search (A) and case-based reasoning (B).

The fact that multiple algorithms from different subfields of AI, based on different formal frameworks can all be executed as sequences of common functions motivates an approach to integrating them. We can create hybrids of two algorithms by interleaving the sequences of common functions that execute each algorithm.

When an algorithm performs a common function on a proposition, we say that it *attends to* or *fixes its attention* on this proposition. We call this event an *attention fixation.* Each of the algorithms described earlier attends to (i.e. executes a common function on) one proposition at a time.

The sequence of attention fixations an algorithm makes when it executes is called its *focus trace*. Focus traces thus provide a uniform way of characterizing the execution of algorithms from different computational methods.

Interleaving the focus traces from two algorithms amounts to executing a hybrid of those algorithms. Specifically, an algorithm H is a hybrid of algorithms A1 and A2 if H's focus trace includes fixations from A1 and A2. Figure 1 illustrates the hybrid execution of case-based reasoning and search. The focus trace (1c) for the hybrid of case-based reasoning and search is a combination of the focus traces for search (1a) and case-based reasoning (1b).

Multiple implementation principle

In each of the examples used to illustrate the common function principle, the common functions were implemented differently. The ability of a surprisingly diverse collection of computational methods to implement each of the common functions is what we call the *multiple implementation principle*. It is a key to enabling the integration of data structures and algorithm described herein. The following examples lend support to the multiple implementation principle.

Forward inference. Both rule-based systems and feed-forward neural networks, though based on very different data structures and algorithms, both take inputs and produce outputs. Forward-chaining in rule-based system matches the left-hand side of a rule against a set of propositions and asserts new propositions. The values of input and output layers of neural networks can be represented as propositions (e.g., a unit representing that the temperature is 25 degrees Celsius can be represented using the proposition *Temperature(25)*). Thus, just like rule-based systems, feed-forward neural networks can be characterized as taking propositions (those representing the values of the input units) and producing new propositions (those representing the values of the output units).

Subgoaling. Backward chaining in rule-based systems, subgoaling in logic-theorem provers and subgoaling in means-end planners are obvious instances of the subgoaling common function. Less obviously, to learn the truth values of propositions representing the output units of neural networks, one can perform a kind of subgoaling by finding (the truth values of propositions representing) the values of the input units. Even directing visual attention is a form of subgoaling. In this case, to learn about the truth value of a proposition representing the attribute of an object, a goal is made of directing a camera or a pair of eyes towards that object.

Identity matching. Algorithms used in object recognition (e.g., Bayesian Networks, neural networks, support-vector machines) all are performing a kind of similarity matching by taking a visually perceived object and finding the most similar stored object. Case-indexing and retrieval schemes in case-based reasoning have the same function.

Representing alternate worlds. Mechanisms which can make inferences about the real world can make inferences about hypothetical worlds.

These examples illustrate that common functions can be implemented using computational methods from very different subfields of artificial intelligence.

Implementing a single common function with multiple different algorithms and data structures enables another kind of hybridization. Consider an algorithm, `A`, executed in a particular instance using common functions `C1 ... Cn`. If each of the `C` are executed using multiple computational methods `M1 ... Mn`, then every step of the execution of algorithm `A` will also involve the methods `M`. For example, one could imagine MCMC executed using a sequence of common functions, each of which is executed by a neural network. Thus, although the neural network would be performing all the computation (in this case computing the likelihood of a proposition given its Markov blanket), the object or focus of that computation can be selected in such a way that it implements MCMC. If one of the `M` executing a common function is an algorithm processing new sensor information, then every step of inference executing using this common function would incorporate up-to-date information from the world.

Cognitive self-regulation

The common function principle shows that very different kinds of algorithms can be executed as sequences of the same set of common functions. How does an intelligent system implementing more than one algorithm decide which common functions to choose? A recurrent theme among the algorithms used to illustrate the common function principle is that they choose which common function to execute in response to *metacognitive problems.* This is evident in several broad classes of algorithms.

Search algorithms choose possible worlds by assuming the truth of propositions that are *unknown*. If the truth value of a proposition is known, it is considered fixed and worlds with that proposition having the opposite value are not explored. In the case of constraint-based search algorithms the *metacognitive* problem is *ignorance*; the truth value of the proposition is not known. In the case of planning-based search, the ignorance is often based on *conflict* since there is more than one possible next action to take or explore.

Many stochastic simulation algorithms also choose worlds to explore based on unknown values of a state variable, but the level of ignorance is somewhat reduced since a probability distribution for the state variable is known. This leads to a somewhat different strategy of exploring worlds: worlds with more likely propositions being true are likely to be explored more often.

Case-based reasoning algorithms are also often driven by ignorance and differ from search and stochastic simulation algorithms by how they react to the ignorance. Instead of exploring possible worlds where different actions are taken, they retrieve similar solutions to similar problems and try to adapt them.

Thus, case-based reasoning, search and stochastic simulation are different ways of addressing different kinds of *metacognitive problems*. We call the insight that many algorithms can be characterized by which common functions they choose in response to metacognitive problems the *cognitive self-regulation principle*.

This analysis resembles Soar's implementation of universal weak methods by reacting to impasses (analogous to metacognitive problems) by reasoning in problem spaces (analogous to alternate worlds). However, together with the common function and multiple implementation principles, the cognitive self-regulation principle motivates a significantly different architecture.

Polyscheme

The principles from the last section motivate the design of a cognitive architecture called Polyscheme. In Polyscheme, algorithms can be implemented as strategies for focusing the attention of multiple modules. By combining strategies, we lay down the hybrid focus traces as described in the last section and thus execute hybrids of algorithms implemented as focus controls strategies. By using modules based on different data structures and algorithms, we enable hybrids of algorithms implemented as focus control strategies and the algorithms in the modules. By including modules that can sense the world and by tightly bounding the time each focus of attention takes, inference can constantly use and react to the latest information from the world. This section and the next elaborate these points.

A Polyscheme system is comprised of a set of specialist, `S`, an attention buffer, `A`, and a focus manager `FM`. FM implements a `getNextFocus()` procedure that returns the next proposition to attend to. All the specialists must implement the following procedures: `Store()`, `OpinionOn()`, `Match()` and `ModifyFM()`. The latter influences which propositions FM selects.

At every time step, FM chooses a proposition for all the specialists to "focus on". Specifically, specialists give their opinions on the proposition's truth value, get all the other specialist's opinions on it, make their own inferences based on this new information and request other propositions for the FM to focus on. The reason all specialists focus on the same proposition at the same time is so that 1. no specialist makes inferences based on a truth value another specialist knows to be incorrect and 2. because even though a proposition is focused on because specialist S1 requested focus on it, it might become relevant for some other specialist S2.

More formally, the Polyscheme control loop is:

```
Do forever:
  A = FM.getNextFocus().
  For all pairs of specialists (S1, S2)
    S1.store(P,S2.opinionOn(A))
  For all S
    S.modifyFM(A)
```

Since once a focus of attention is chosen, all the specialists must focus on and therefore execute the same functions on it, the FM controls the flow of computation. We have experimented with several kinds of FMs, but will use a very simple FM based on a queue (FQ) to illustrate how attention selection can implement hybrids of many kinds of algorithms. Although achieving all our long-term objectives will almost certainly require a more sophisticated focus manager, a queue-based focus manager has been sufficient to achieve significant results and demonstrate the promise of the approach.

Algorithms as focus control strategies

In this subsection, we show how to implement algorithms using focus management in Polyscheme and illustrate the integration this enables.

First, let us consider how to implement a local search algorithm such as WalkSAT. We can implement pure WalkSAT using a "constraint specialist". At any given time this specialist encodes a constraint C in conjunctive normal form. The specialist's functions operate as follows:

`Store(P,TV)`: `P` and TV are added as a clause to `C`.

`ModifyFM(P)`: If C is satisfied in `P`'s world, w, then add `Satisfied(C, E, w)` to FQ. Otherwise, return the proposition `P` that leads to the most clauses in C becoming satisfied.

`OpinionOn(P)`: If `P` is of the form `Satisfied(C,w)` return `T` if `C` is satisfied in `w`, otherwise return `U`.

A system with repeated experience in similar environments will have made many inferences and solved many problems. One can speed it up using a form of case-based reasoning. A "case specialist" remembers previous states of the worlds and the relations among objects in those situations. In a new situation, it is capable of finding similarities with the past and suggesting cases relevant to the present. (The particular similarity metric is not relevant to how case-based reasoning fits into the system.) More specifically:

`Store(P,TV).` `(P,TV)` is added to memory.

`ModifyFM(P).` If `P` is involved in a structure (i.e., a set of related propositions) that is highly similar (above some threshold) to a to a proposition `P1` in a previously encountered structure, then transform the old case into propositions that use the objects and times in the current situation and put those proposition on the queue.

Case-based reasoning and WalkSAT integrate easily. WalkSAT proceeds as it normally would, flipping the truth value of one proposition at time and focusing on it. When the case specialist finds a situation in the past that is sufficiently similar to this one, it essentially flips several truth values at once, hopefully getting WalkSAT much closer to a solution. The brittleness of CBR is ameliorated by the fact that whatever inconsistencies there are between the past case and the current situation can be resolved by continued search by WalkSAT.

Finally, for environments that are changing and/or we can only perceive through noisy sensors, every step of case-based reasoning and WalkSAT should be influenced by information from these sensors.

Adding a "perception specialist" achieves this:

`ModifyFM(P).` If sensors perceive that a proposition `X` is true or false, put `X` on the beginning of `FQ`.

`OpinionOn(P).` Returns the truth value `P` had the last time it perceived it or `U` if you have never perceived it.

Thus, during any attention fixation caused by the constraint or case specialist, if new information about a proposition `P` is perceived, the perception specialist will ask the focus manger to focus on it. When Polyscheme does focus on `P`, the perception specialist will take the appropriate stance on it. The case and constraint specialists will then learn about it through their `Store()` procedures and incorporate the information into their case-matching and search, respectively.

These examples illustrate that algorithms from very different computational frameworks can be integrated using the same "computational building blocks", i.e., the focus of attention of a set of modules.

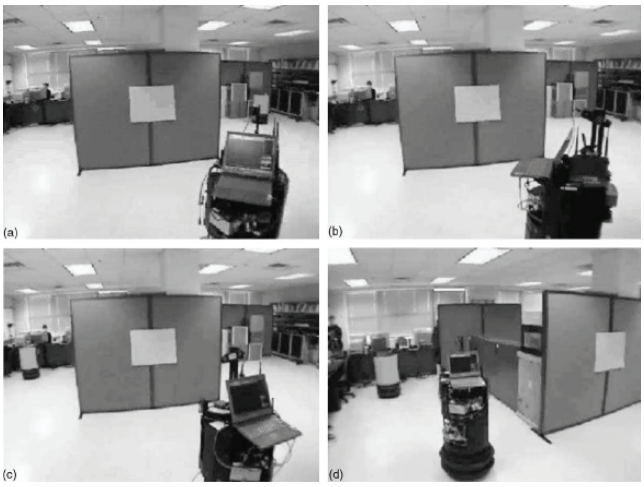

Figure 2. The robot tracking problem. The robot in the foreground of (a) is tasked to track the robot in the distant right of (a). The tracked robot disappears (b) behind an occluder and a robot with the exact appearance emerges (c) from the left of the occluder. Polyscheme infers the two are identical and moves the tracking robot towards the visible robot. When (d) it sees an obstacle to robot motion behind the occluder, it infers the two robots are different.

The benefits of focus trace integration

We will describe how Polyscheme enables the best characteristics of multiple diverse computational methods to be combined by describing a working mobile robot (N. L. Cassimatis, Trafton, Bugajska, & Schultz, 2004) controlled by Polyscheme. The robot's goal (illustrated in Figure 2) was to keep track of and follow another robot as it moved about and occasionally became occluded by other objects. This task requires many of the characteristics of intelligent systems we described in the introduction. The reasoning and planning problem was difficult. Because actions of robots had side effects and because there was incomplete information (because of occlusion), traditional planners could not be used. Formulating the physical constraints on objects (e.g., they do not pass through each other, gravity, etc.) in this domain in a SAT solver was prohibitive. Idealizing the robot's physical environment into a 5x5x5 grid exhausted the memory of a computer running no other applications. Further, although operating in an environment while taking into account physical laws and potentially occluded objects is well beyond the reach of pure reactive systems, the positive characteristics of these systems were required. The environment was constantly changing, the robot had incomplete information and it therefore needed to be reactive and flexible enough to incorporate new information from its sensors into its planning and inference.

We briefly describe the Polyscheme system that controlled the robot. The focus manager was a modification of the queue scheme described above. The main modification was that propositions in the queue were associated with "satisfaction conditions" that would cause the proposition to be removed from the queue when they were met. For example, if proposition `P1` was put on the queue to help infer if `P2` was true, if the truth of `P2` is determined, there is no longer a need to focus on `P1` and it is removed. The specialists included a perception specialist that detected the location and type of objects in the environment. A physical constraint specialist kept track of physical constraints and a path specialist included a library of "path scripts" that described paths robots typically take.

We can now use this robot to help describe how Polyscheme enables systems that exhibit the characteristics of algorithms based on diverse computational formalisms.

Generality and flexibility. Methods such as local search, backtracking search and stochastic simulation make inference, find plans or solve constraints in a very wide variety of domains. This makes them general and flexible in that when a situation changes and is formulated for these algorithms, they will deal with them accordingly. We have already described how to implement such algorithms within Polyscheme. In our robot, these kinds of algorithms were used to maintain the physical constraints described above. The benefit of doing so is that the system can also exhibit the following characteristics, which not often exhibited in pure versions of these general algorithms operating on many classes of problems.

Speed. General algorithms tend to be slow on larger problems because the state spaces they explore grow very quickly as a problem grows. "Structured" reasoning and

planning algorithms based on frames or scripts do not have this problem since they do not generally search state spaces but instead make inferences or solve problems using large structured representations (i.e., frames, scripts or cases). When these algorithms are implemented in Polyscheme together with more general and flexible algorithms, the best characteristics of each can be exhibited in the same system. In our robot, Polyscheme could find a continuous path between two sightings of a tracked object more quickly than pure search because of its library of path scripts. When a script was retrieved that did not completely match the existing situation, the constraint system would detect this contradiction and this would initiate a search for a model for changes to the script that would be more consistent with the situation. While full search was always available, the script retrieval effectively meant that search began in a state much closer to the correct model of the world. Thus, the speed of structured approaches was combined with the generality and flexibility of search-based methods.

Reactivity. Environments whose states change and which are sensed through imperfect sensors require systems to be able to quickly update their plans and inferences upon new information. Reactivity can be achieved using Polyscheme by mandating each focus of attention be quick and by including sensor specialists. This guarantees that little inference will happen before new information is sensed. When our robot moved to a different location and saw objects that were formerly occluded, the perception specialist requested that all the other specialists focus on this information immediately. When they did, the constraint and path specialist were able to update their inferences and plans accordingly.

The ability of the robot to engage in complex reasoning and planning while moving about in a changing world demonstrates that creating hybrids of algorithms in Polyscheme can enable a combination of characteristics not possible given existing individual methods alone.

Evaluations and implemented systems

The goal of this paper is to present a cognitive architecture for generating hybrids of algorithms that have capabilities impossible to achieve by those algorithms individually or in some purely modular combination. The principle way to demonstrate that it can do this is to actually build such systems. The robot from the last section is a principle example. It can react to changes while making inference and finding plan that purely reactive systems cannot and which literally crashed computers when conventional inference and reasoning algorithms worked on them (as we describe in the SAT example above).

Polyscheme can also be used to integrate general probabilistic inference with probabilistic context-free parsing so that sentences can be disambiguated using world knowledge. PCFGs are too restricted to formalize many important concepts, but general probabilistic inference approaches cannot parse sentences because they require all possible parses to have already been enumerated before they can even operate. The key here was to integrate MaxWalkSAT-like local search to maximize the probability of a parse together with production rule based methods for creating potential parses.

A heterogeneous database retrieval system (Nicholas L. Cassimatis, 2003) that makes sound and complete inferences has also been implemented within Polyscheme. It implements resolution theorem proving using common functions such as forward inference, subgoaling, and identity. Each of these operations is implemented in specialists based on representations such as neural networks, production rules, geospatial coordinates and relational databases. If these modules meet certain conditions (which in practice are easy to confirm), the total system's inference is sound and complete. This system demonstrates how hybridizing algorithms provides the benefits of both logic programming (provable soundness and completeness) and the efficiency of special representations (such as relational databases).

Each of the systems above solve problems that could not be solved by existing computational methods alone and therefore demonstrate that executing hybrid algorithm through a focus of attention in Polyscheme can enable advances in computational intelligence.

Other approaches to integration

Most approaches to integrating algorithms and/or their characteristics into a single system have taken a reductive, modular or "fixed" hybrid approach. Reductive approaches tend to implement an algorithm or solve problems by a reduction to another approach. This tends to consign such systems to the limitations of the computational formalism or method being reduced to. For example, reducing a first-order probabilistic logic reasoning problem to a graphical model belief propagation problem (Domingos & Richardson, 2006), means that one is still limited by the "closed-world" and scalability characteristics of belief propagation inference algorithms. Modular architectures for integration tend to enable modules based on different data structures and algorithms to communicate and/or cooperate. Normally the only way to add an algorithm to such a system is to add a module based on that algorithm. Polyscheme enables this kind – there are parallels in of integration (Schubert, Papalaskaris, & Taugher, 1983) -- but also enables algorithms to be executed through the focus of attention. This allows every single step of every algorithm to be executed using many data structures and algorithms and thus enables a much more thorough integration of algorithms. Fixed hybrid approaches such as Clarion (Sun, 2004) and ACT-R (Anderson, 2005; Anderson & Lebiere, 1998) create hybrids between a few specific algorithms. Both Clarion and ACT-R are production systems that use reinforcement-learning mechanisms for conflict resolution, but do not enable such close interaction between other algorithms implemented in those systems.

Polyscheme differs from many cognitive architectures in being primarily inferential and not procedural. Most architectures, e.g., Icarus (Langley & Choi, 2006), Soar (Laird, Newell, & Rosenbloom, 1987), Epic (Kieras & Meyer, 1997), Clarion and ACT-R choose an action at every step. These actions are actual physical actions or operations on data structures in memory. In Polyscheme, at every time step, specialists do not take or propose actions, but instead take stances on the truth value of propositions and suggest propositions to attend to. A consequence is that Polyscheme includes mechanisms for communication and sharing information about the truth of propositions that make it much easier to implement many reasoning and inference algorithms.

A focus of attention is a very important part of ACT-R and Rao's work (Rao, 1998) on visual routines. However, in ACT-R, the focus of attention is not used to implement algorithms and in neither case is it feasible (because of the absence of mechanisms involving truth values and alternate worlds) to implement and integrate reasoning and inference algorithms using this sort of focus of attention mechanism.

Finally, Polyscheme's focus of attention is superficially reminiscent of blackboard systems insofar as it is a shared entity multiple modules access. However, the two have numerous and profound differences: the focus of attention in Polyscheme is tiny; Polyscheme has no shared-memory among modules while blackboards are such shared memory; Polyscheme modules can have arbitrarily large, varied and persistent memories while in many blackboard systems, the blackboard is the main form of memory; blackboard systems have no straightforward way of exploring alternate states of the world; and unlike the focus of attention in Polyscheme, modules in blackboard systems generally do not synchronize their operation.

Conclusions

Our goal is to create a single system that can exhibit the best characteristics of algorithms based on different computational formalisms. Our approach has been to create a cognitive architecture called Polyscheme that can execute hybrids of these algorithms. Polyscheme enables two kinds of hybrids. First, a Polyscheme system can include modules based on arbitrarily different algorithms and data structures. Second, and most originally, Polyscheme can execute algorithms by guiding the "focus of attention" of these modules. Systems created using Polyscheme demonstrate that integration through a focus of attention enables systems that can make inferences and solve problems in situation beyond the reach of existing individual computational methods.

References

Anderson, J. R. (2005). Human symbol manipulation within an integrated cognitive architecture. *Cognitive Science*, 313-341.

Anderson, J. R., & Lebiere, C. (1998). *The Atomic Components of Thought*. Hillsdale, NJ: Lawrence Erlbaum Associates.

Cassimatis, N. L. (2003). *A Framework for Answering Queries using Multiple Representation and Inference Technique.* Paper presented at the 10th International Workshop on Knowledge Representation meets Databases.

Cassimatis, N. L., Trafton, J. G., Bugajska, M., & Schultz, A. C. (2004). Integrating Cognition, Perception, and Action through Mental Simulation in Robots. *Robotics and Autonomous Systems, 49*(1-2), 13-23.

Domingos, P., & Richardson, M. (2006). Markov Logic Networks. *Machine Learning, 62*, 107-136.

Kieras, D., & Meyer, D. E. (1997). An overview of the EPIC architecture for cognition and performance with application to human-computer interaction. Human-Computer Interaction. *12*, 291-438.

Kolodner, J. (1993). *Case-Based Reasoning*. Menlo Park, CA: Morgan Kaufman.

Laird, J. E., Newell, A., & Rosenbloom, P. S. (1987). Soar: An architecture for general intelligence. *Artificial Intelligence, 33*, 1-64.

Langley, P., & Choi, D. (2006). *A unified cognitive architecture for physical agents.* Paper presented at the Twenty-First National Conference on Artificial Intelligence, Boston.

Minsky, M. L. (1975). A Framework for Representing Knowledge. In P. H. Winston (Ed.), *The Psychology of Computer Vision*. New York, NY: McGraw -Hill.

Rao, S. (1998). *Visual Routines and Attention.* Unpublished Doctoral, Massachusetts Institute of Technology, Cambridge, MA.

Schank, R. C., & Abelson, R. P. (1977). *Scripts, plans, goals, and understanding: An inquiry into human knowledge structures*. Hillsdale, NJ: Lawrence Erlbaum.

Schubert, L. K., Papalaskaris, M. A., & Taugher, J. (1983). Determining Type, Part, Color and Time Relationships. *IEEE Computer, 16*(10), 53-60.

Selman, B., Levesque, H., & Mitchell, D. (1992). *A new method for solving hard satisfiability problems.* Paper presented at the Proceedings of the Tenth National Conference on Artificial Intelligence.

Sun, R. (2004). The CLARION cognitive architecture: Extending cognitive modeling to social simulation. In *Cognition and Multi-Agent Interaction*. New York, NY: Cambridge University Press.