Rethinking Cognitive Architecture via Graphical Models

Paul S. Rosenbloom
Department of Computer Science and Institute for Creative Technologies
13274 Fiji Way, Marina del Rey, CA 90292 USA
Email: rosenbloom@usc.edu; Phone: (310) 448-5341; Fax: (310) 301-5009

Abstract

Cognitive architectures need to resolve the *diversity dilemma* – i.e., to blend diversity and simplicity – in order to couple functionality and efficiency with integrability, extensibility and maintainability. Building diverse architectures upon a uniform implementation level of graphical models is an intriguing approach because of the homogeneous manner in which such models produce state-of-the-art algorithms spanning symbol, probability and signal processing. To explore this approach a *hybrid* (discrete and continuous) *mixed* (Boolean and Bayesian) version of the Soar architecture is being implemented via graphical models. Initial steps reported here begin to show the potential of such an approach for cognitive architecture.

Keywords: Cognitive architecture; graphical models; factor graphs; production match; Soar; Markov logic

A *cognitive architecture* seeks to provide a coherent integration of capabilities sufficient for human-level artificial intelligence, whether as a detailed model of human cognition or a system more loosely tied to the specifics of human behavior. Such an architecture requires the integration of a wide range of cognitive capabilities for, among other things, representation and memory, problem solving and planning, learning, reflection, interaction (including perception and motor control, use of language, etc), and the social aspects of cognition (such as emotion, collaboration, etc.).

A key issue with such architectures is what can be called the *diversity dilemma*: they must simultaneously be both diverse *and* simple. Diversity of functionality is required to support intelligent behavior in a complex and uncertain world and to model the range of experimental results exhibited by human cognition. As an architectural strategy, diversity implies openness to a proliferation of mechanisms, and appeals to notions of specialization and optimization. In caricature it is the biologist's approach, in which many specialized structures, each locally optimized, jointly yield a robust and coherent whole. Simplicity on the other hand is critical for architectural integrability, extensibility and maintainability. As an architectural strategy, it implies conservatism towards the addition of mechanisms, and appeals to notions of elegance and uniformity. In caricature it is the physicist's approach, where a broad diversity of phenomena emerges from interactions among a small set of general elements.

Traditionally architectures have had to choose one approach or the other. Among architectures for cognitive modeling, Soar [1] has been a canonical exemplar of simplicity/uniformity and ACT-R [2] of diversity. However, Soar 9 [3] has recently shifted strongly towards diversity. Looking back, the history of Soar [4,3] could be said to illustrate a *uniformity-first* research strategy (a variant of *Ockham's razor*): begin by assuming uniformity and accept diversity only upon overwhelming evidence. To the extent uniformity is possible, such a strategy yields elegance and facilitates unification and extension. Beginning instead with diversity removes the pressure to search for hidden commonality, and may lead down an irrevocable path of complexity. For years Soar had a single procedural, rule-based, long-term

memory and a single learning mechanism, while investigations were pursued into the ability of this combination to support a diversity of memory (e.g., procedural, semantic, and episodic [5]) and learning (e.g., skill and knowledge acquisition, generalization and transfer, and learning from observation [6]) behaviors. A wide range of such behaviors proved feasible, but they never could be fully unified with the rest of the system to yield pervasive utility across all activity. This evidence against the existing uniformity, amassed over years of experimentation, inspired the development of Soar 9, a diverse architecture that adds new long-term memories (semantic and episodic) and learning mechanisms (semantic, episodic and reinforcement), while also incorporating other new capabilities (emotion and imagery).

But the acceptance of architectural diversity leaves open the question of how to deal with issues of integrability, extensibility and maintainability; not to mention elegance, to the extent it is considered a relevant factor. Diverse architectures, such as Soar 9 and the various flavors of ACT, have historically been difficult to integrate, often exhibiting more the character of a toolkit than an architecture. Software engineering techniques, such as modularization, provide one path for coping with such issues. Cognitive modeling frameworks such as Storm [7] and Cogent [8] are, for example, exploring this path. But there is an alternative path, based on continued adherence to uniformity-first: combining an acceptance of the need for diversity in the architecture with a continued search for uniformity elsewhere in cognition.

Newell [9] proposed understanding cognition in terms of a hierarchy of spatiotemporal levels. Given such a hierarchy, it is possible to ask about the *girth* – that is, the variety of mechanisms – at each level. Diversity always exists across levels, but individual levels may consist of anything from a small number of very general elements to a wide diversity of more specialized ones. Across a hierarchy of levels, there is no a priori reason to assume they are all of comparable girth. While physicists and biologists may expect uniformity within their fields, the networking community trumpets the *Internet hourglass* to explain their protocol stack [10]. At the narrowed waist is the Internet Protocol (IP). Above is an increasingly diverse sequence of levels enabling "everything on IP". Below is an increasingly diverse sequence of levels enabling "IP on everything". The hourglass yields a diversity of applications and implementations that are united via a core of *mesoscale uniformity*.

In analogy, we can ask whether there remains a core of mesoscale uniformity in cognition, and whether it can be the key to resolving the diversity dilemma. There must clearly be diversity at the top of the cognitive hierarchy; the extraordinary range of behaviors and applications of which humans are capable is one of the core phenomena cognitive architectures are developed to explain. At the bottom, the mind is grounded in the diverse biology of the brain and, at least according to *strong AI*, could also be grounded in a diversity of alternative technologies. But is there an hourglass or a rectangle in between? Domingos's recent call for an *interface layer* in AI [11] amounts to an appeal for a *cognitive hourglass* in the building of AI systems in general, although not directly focused on architecture. Within cognitive modeling, the question of the existence of a cognitive hourglass has historically been cast in terms of whether the cognitive architecture is uniform. The approach here is instead to accept the need for diversity in the architecture, as in ACT-R and Soar 9, while continuing the search for uniformity and simplicity at the *implementation level* below the architecture. The goal is still an hourglass, albeit one with a lower waistline.

Architectural implementation has traditionally been considered mere "implementation details," of pragmatic importance for efficiency and robustness, but of little theoretical interest. The one notable exception has been when a symbolic architecture is implemented via neural

networks, as in Neuro-Soar [12] and SAL [13]. The approach here is to focus instead on the specific class of *graphical models* [14] that arise from the decomposition of multivariate functions. Neural networks are clearly graphical, but may or may not be graphical models in this more specific sense. Bayesian networks [15] are directed graphical models over random variables that have revolutionized probabilistic reasoning. Markov networks are undirected analogues of Bayesian networks that go beyond probabilities to allow general weights, or *clique potentials*, on groups of variables. Factor graphs [16] are undirected, like Markov networks, but were developed in coding theory – where they underlie the "astonishing performance" of turbo codes – to represent and reason efficiently about general multivariate functions. They differ from Markov networks in replacing clique potentials with factor nodes directly in the graph.

One of the most intriguing aspects of graphical models from an architectural perspective is their ability to uniformly process symbols, probabilities and signals via variants of the same graph representation and inference algorithm (the *sum-product algorithm*). This approach is not only uniform, but it subsumes state-of-the-art algorithms spanning these areas, such as arc consistency (symbol processing), loopy belief propagation (probability processing), and Kalman filters and the forward-backward algorithm in HMMs (signal processing). It has also been shown that a variety of neural network algorithms – such as supervised Boltzmann machines, radial basis functions, and unsupervised learning algorithms – can be mapped onto graphical models [17]. Graphical models provide a tantalizing combination that is particularly appropriate at the implementation level of: (1) *generality* in the range of capabilities they can uniformly support in a state-of-the-art manner; and (2) *constraint* in the ways that these capabilities can reasonably be supported.

If these capabilities can all be leveraged in a unified fashion, they promise broadly functional *hybrid* (combining both discrete and continuous processing) *mixed* (combining both Boolean and Bayesian reasoning) architectures that are grounded in a simple uniform implementation level. Such an approach could elegantly explain the diversity seen in existing cognitive architectures while going beyond them to yield an effective and uniform basis for: unifying cognition with perception and motor control by breaking down the barriers between central and peripheral processing, bringing the latter within the cognitive inner loop and making each form of processing potentially penetrable by the other; fusing symbolic and probabilistic reasoning to provide general reasoning under uncertainty; and providing a conceptual bridge from symbolic to neural architectures, by mapping them onto a common intermediary. Existing work on hybrid mixed methods is encouraging [18], as is work on general languages for mixed probabilistic and logical reasoning. FACTORIE [19], for example, combines factor graphs with an imperative programming language to support relations and other capabilities, while BLOG [20] and Alchemy [21] combine probability and logic via Bayesian and Markov networks, respectively.

The particular approach advanced here is to: (1) re-implement existing architectures to help understand them and the implications of implementing them via graphical models; (2) go beyond existing architectures by hybridization and simplification, both within and across architectures; (3) integrate in new capabilities that don't mesh well with existing architectures, such as perception and motor control; and (4) explore radically new architectures enabled by the unique strengths of graphical models. Ultimately the hope is for both a better understanding of the space of architectures and the development of radically new architectures embodying heretofore-unseen combinations of functionality and simplicity.

The initial focus is on a graphical reimplementation and extension of Soar, with the aim of developing a uniformly implemented mixed hybrid variant. Soar is particularly useful as a

starting point because it: is one of the longest standing – over 25 years – and most thoroughly investigated cognitive architectures; has been explored as both a unified theory of human cognition and as an architecture for intelligent agents and virtual humans; and exists in both uniform (versions 1-8) and diverse forms (version 9), enabling a strategy of starting reimplementation with the initial uniformity while seeking opportunities for a more uniform integration of the later diversity. Concurrently we can explore extensions beyond Soar's predominant symbol processing paradigm, through the deep integration of probability and signal processing in support of improved reasoning about, and interaction with, the real world.

This article reports the first steps along this path. Section 1 presents the essence of "uniform" Soar (versions 3-8) across a hierarchy of three cognitive levels (at time scales of 10ms-1s). Section 2 provides an introduction to factor graphs as an exemplar of graphical models, and as the key technology for a graphical reimplementation of Soar's lowest level, the *elaboration cycle*. This reimplementation, which is described in Section 3, establishes the initial applicability of graphical models to production match, a central component in Soar and other architectures. Section 4 then explores the next level up in Soar, the *decision cycle*, using Alchemy to derive insights into what will be necessary for a mixed variant that is capable of general symbolic reasoning under uncertainty. Section 5 concludes with next steps.

1 Cognitive Scales and Soar

Newell's notion that scale counts in cognition provides a key component of Soar's theoretical background as a model of human cognition. The core idea is that the phenomena of interest in cognition change as the focus shifts from small spatiotemporal scales to larger ones. Newell discusses time scales from 10^{-4} sec (100 μs) up to 10^{7} sec (months), stratifying them into four bands in human cognition: biological (10^{-4}-10^{-2} sec), cognitive (10^{-1}-10^{1} sec), rational (10^{2}-10^{4} sec) and social (10^{5}-10^{7} sec). In the biological band in particular there is also a spatial aspect to these scales, since signals within the brain are limited in how far they can travel within such small time periods. Organelles (10^{-4} sec), neurons (10^{-3} sec) and neural circuits (10^{-2} sec) yield spatial scales within the biological band, before primitive deliberate acts (10^{-1} sec) and operations (10^{0} sec) are reached at the base of the cognitive band.

Architectural mechanisms in uniform versions of Soar were traditionally mapped onto a subset of these time scales, as in Figure 1, starting with the *elaboration cycle* at 10 ms (neural circuits), the *decision cycle* at 100 ms (deliberate acts), and *problem space activity* at 1 sec (operations). The elaboration cycle involves parallel match – via a variant of the Rete algorithm [22] – and firing of productions based on the contents of a global working memory (WM). Functionally, it achieves one round of parallel associative retrieval of information relevant to the current situation. Production actions specify knowledge for retrieval while production conditions specify when it should be retrieved. Conditions also bind variables for use in actions.

The decision cycle begins with repeated cycles of elaboration until quiescence; i.e., until no more productions can fire. This *elaboration phase* is followed by a decision based on preferences retrieved during elaboration. The elaboration phase yields an interpretation of the current situation, while the decision either

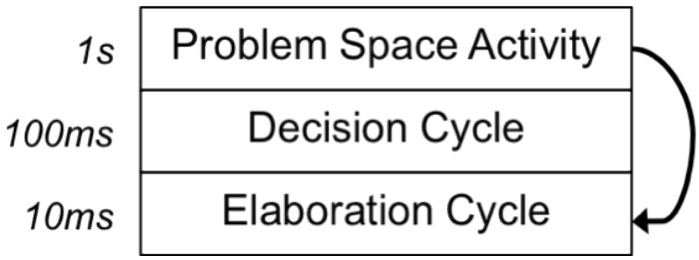


Figure 1: Soar's architectural levels (with chunking)

selects an operator to apply or generates an *impasse* if no operator can be selected. Impasses engender *reflection*, enabling processing to recur at the meta-level on the problem of making the decision. The decision cycle is Soar's cognitive inner loop; it accesses whatever knowledge is immediately available about the current situation and then attempts to decide what to do next.

A sequence of decisions yields activity in a problem space, amounting to some form of search if knowledge is limited and impasses occur. Search in problem spaces (ps-search) is: *slow*, with each decision occurring at the 100 ms level; *serial*, via a sequence of operator selections and applications; and potentially *combinatoric*, yielding trees that grow exponentially in the depth of the search. However, ps-search is open to control by any knowledge accessible during the decisions that occur as part of it. When the knowledge is sufficient to uniquely determine the outcome of each decision, behavior is more accurately characterized as algorithmic, or knowledge-driven, than as search.

Determining which knowledge to access during a decision can also be viewed as a search process – termed knowledge search (k-search) – but one that contrasts strongly with ps-search in character. K-search is: *fast*, with a 10 ms cycle time, *parallel*, both in match and firing of productions; and *subexponential*, at least in theory, if not in reality in most implementations. K-search occurs over a closed, extensionally defined, set of structures – the knowledge/productions in the system – rather than dynamically generating an open search space in the manner of ps-search. It is inherently algorithmic, rather than using an open cognitive loop, and is thus not itself penetrable by additional control knowledge.

Chunking [23] is a learning mechanism in Soar that generates new productions based on the results of problem space activity during impasses. It compiles knowledge that is initially only available through activity at time scales of 1 second or more down to knowledge that is "immediately available" for use at the 10 ms time scale. It automatizes behavior [24] while speeding up overall performance. Although chunking, in combination with the flexibility of Soar's problem solving, has been shown to yield a much wider range of learning behaviors than just automatization [6] – such as concept acquisition and episodic learning – speeding up behavior remains its most essential functionality. Responsibility for most of these other learning activities has been taken over by Soar 9's new learning mechanisms.

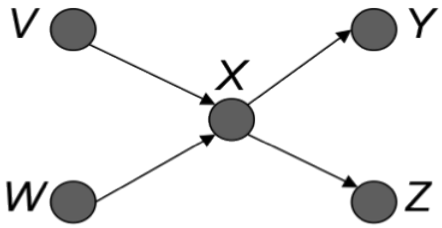
2 Factor Graphs

Factor graphs provide a form of divide and conquer with nearly decomposable components for reducing the combinatorics that arise with functions of multiple variables. The function could be a joint probability distribution over a set of random variables; e.g., $\mathbf{P}(V,W,X,Y,Z)$, which yields the probability of $V=v \wedge W=w \wedge X=x \wedge Y=y \wedge Z=z$ for every value v, w, x, y and z in the variables' domains. Or the function could represent a constraint satisfaction problem – e.g., $\mathbf{C}(A,B,C,D)$ – over a set of variables, yielding 1 if a combination of values satisfies the constraints and 0 otherwise. Or the function could represent a discrete-time linear dynamical system, as might typically be solved via a Kalman filter. The problem formulation here would involve a *trellis* structure, where the graph for one time step is repeated for each, with four variables per time step (*State*, *Input*, *Output* and *Noise*) [16]: $\mathbf{K}(S_0,I_0,O_0,N_0,\ldots, S_n,I_n,O_n,N_n)$.

The prototypical factor graph operation is the computation of *marginals* on variables. For a joint probability distribution, this is simply the marginal distribution of a random variable, as computed by summing out the other variables; e.g., $\mathbf{P}(Y) = \Sigma_{v,w,x,z}\mathbf{P}(v,w,x,Y,z)$. The key to tractability is avoiding the explicit examination of every element of the cross product of the variables' domains. For probabilities, the joint distribution is decomposed into a product of

conditional and prior probabilities over subsets of variables; e.g., $\mathbf{P}(V,W,X,Y,Z) = \mathbf{P}(V)\mathbf{P}(W)\mathbf{P}(X|V,W)\mathbf{P}(Y|X)\mathbf{P}(Z|X)$. Such decompositions derive from the *chain rule* plus *conditional independence* assumptions. Commutative and distributive laws then improve efficiency by enabling variables to be summed out as soon as possible – e.g., $\mathbf{P}(Y) = \Sigma_x\mathbf{P}(Y|x)\Sigma_z\mathbf{P}(z|x)\Sigma_v\mathbf{P}(v)\Sigma_w\mathbf{P}(x|v,w)\mathbf{P}(w)$ – yielding the basis of Bayesian networks (Figure 2).

Factor graphs generalize this to arbitrary multivariate functions; e.g., $\mathbf{F}(V,W,X,Y,Z) = \mathbf{F}_1(V,W,X)\mathbf{F}_2(X,Y,Z)\mathbf{F}_3(Z)$. The function becomes a bipartite graph with a *variable node* for each variable, a *factor node* for each subfunction, and undirected links between factors and their variables (Figure 3).



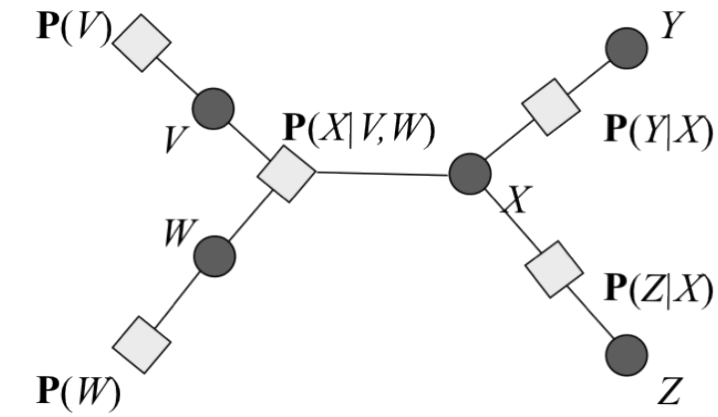
$$\mathbf{P}(V,W,X,Y,Z) = \mathbf{P}(V)\mathbf{P}(W)\mathbf{P}(X|V,W)\mathbf{P}(Y|X)\mathbf{P}(Z|X)$$

Figure 2. Sample Bayesian network

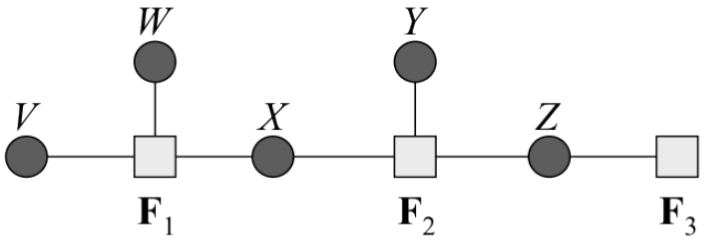
The core inference algorithm for factor graphs is the *sum-product* (aka *summary-product* or *belief-propagation)* algorithm, which passes messages along links. A message from a source node to a target node along a link is a vector that summarizes the source node's information about the domain of the link's variable node; e.g., a probability distribution over the variable's domain elements. A message from a variable node to a factor node is the *pointwise product* of the messages into the variable node from all of its neighbors except the target node. A point-wise product is akin to an inner product, where the corresponding values from the two vectors are multiplied, but with the individual products then forming a new vector rather than being summed into a single value.

A message from a factor node to a variable node starts with this same pointwise product but also includes the factor node's own function in the product. All variables other than the target variable are then summed out to form the factor's outgoing message. A key optimization here, as in Bayesian networks, is to use the commutative and distributive laws to redistribute multiplicative factors outside of summations.

For tree-structured factor graphs in which only a single marginal is desired, the graph can be reduced to an *expression tree* in which the products and sums are computed upwards towards the root of the tree. Beyond this simplest case, the



(a) $\mathbf{P}(V,W,X,Y,Z) = \mathbf{P}(V)\mathbf{P}(W)\mathbf{P}(X|V,W)\mathbf{P}(Y|X)\mathbf{P}(Z|X)$



(b) $\mathbf{F}(V,W,X,Y,X) = \mathbf{F}_1(V,W,X)\mathbf{F}_2(X,Y,Z)\mathbf{F}_3(Z)$

Figure 3. Sample factor graphs

algorithm works iteratively by sending output messages from nodes as they receive input messages. For *polytrees*, which have at most one path between any two nodes, this iterative algorithm always terminates and yields the correct answer. For arbitrary graphs with loops, neither correct answers nor termination are guaranteed. However, it does often work quite well in practice, as has been most strikingly evident for turbo codes.

The sum-product algorithm utilizes two specific arithmetic operations: sum and product. However, the same generic algorithm works for any pair of operations forming a *commutative semi-ring*, where both operations are associative and commutative and have identity elements, and the distributive law exists. Max-product, for example, is key to computing maximum a posteriori (MAP) probabilities. Or-and also works, as do a range of other operation pairs.

To improve the efficiency of the sum-product algorithm, various additional optimizations can also be applied, and alternative algorithms can be used (such as survey propagation [25] and Monte Carlo sampling [26]). A connection exists between factor graphs and statistical mechanics, revealing that the sum-product algorithm minimizes the *Bethe free energy*, and yielding further algorithmic innovations [27].

3 Reimplementing Soar's Elaboration Cycle via Factor Graphs

The core computational process in Soar's elaboration cycle is production match, as implemented by the Rete algortihm. Rete consists of a discrimination network for sorting working memory elements (wmes) to matching production conditions; a join network to determine which combinations of wmes yield production instantiations while respecting across-condition variable equality constraints; and support for both incremental match across cycles and shared match across productions. Most individual productions match efficiently, although worst-case match cost is exponential in the number of conditions.

The purpose of this section is to demonstrate that graphical models in general, and factor graphs in particular, can straightforwardly implement a state-of-the-art production match algorithm – i.e., one that is as good as or better than Rete – and in so doing to expand their known repertoire within symbol processing to a capability that is used in many cognitive architectures and that is absolutely central to Soar. The functionality implemented in this section does not go beyond what already exists in Soar, but it does show how such a capability can be provided uniformly with the other forms of symbol, probability and signal processing that are already part of the graphical model oeuvre.

A mapping of Rete onto factor graphs has been designed on paper in which factor nodes handle discriminations and joins, variable nodes effectively represent both the wmes that match production conditions and the instantiations that match combinations of conditions – analogous to Rete's α and β *memories*, respectively – and unidirectional message passing over an expression tree enables incremental and shared match. However, rather than imposing Rete a priori on factor graphs, the primary focus in these investigations has been on algorithms that arise naturally from viewing production match as a multivariate function.

Consider the rule in Figure 4, for simple color inheritance. This is not exactly Soar's representation, although it does retain Soar's object-attribute-value scheme, with conditions testing wmes via constants and variables (denoted in angle brackets). The simplest

```
P1: Inherit Color
    C1: (<v0> ^type <v1>)
    C2: (<v1> ^color <v2>)
    →
    A1: (<v0> ^color <v2>)
```

Figure 4. Sample color inheritance rule

mapping of this production to a factor graph views it as a Boolean function of the three production variables – $\mathbf{P_1}(v_0,v_1,v_2)$ – which, for each combination of variable values, yields 0 or 1 depending on whether the combination defines a legal instantiation. The production's conditions then specify how the function is to factor: $\mathbf{P_1}(v_0,v_1,v_2) = \mathbf{C_1}(v_0,v_1)\mathbf{C_2}(v_1,v_2)$ (Figure 5).

In the implementation of this mapping, WM is a 3D Boolean array – objects × attributes × values – with 1s for every wme in WM and 0s elsewhere; and messages are Boolean vectors with 1s for valid bindings of the link's variable and 0s elsewhere. In essence, productions define graphs while WM defines distributions over graph variables.

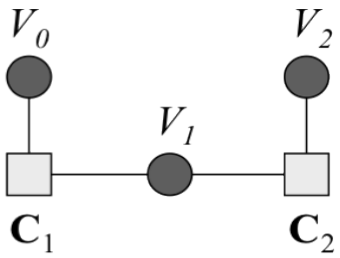


Figure 5. Sample rule graph

This initial approach indicates the feasibility of implementing match via factor graphs, but it also raises three issues: (1) both WM and constant tests are hidden within condition factors; (2) production match ignores conditions without variables in determining legal instantiations; and (3) it leads to errors from *binding confusion* [28]. The first problem doesn't affect correctness – only how much message processing is leveraged during the match process – and the second can't actually occur in Soar because all of the conditions in its productions must be linked via variables, so only brief overviews of their solutions are provided before the third issue is discussed in some detail.

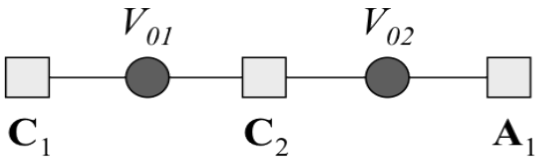
To use message processing more directly in handling WM and constant tests, new factor nodes are added to the graph, with WM and constants being represented explicitly via their functions. By connecting these nodes to the condition factors through new variables and their associated variable nodes, condition match automatically becomes appropriately constrained when the messages from these nodes are processed. For example, in condition `(<v`$_o$`> ^type <v`$_1$`>)` the constant attribute `type` is replaced by a new variable, yielding `(<v`$_o$`> ^<x> <v`$_1$`>)`, with the new variable being constrained by a factor node whose function is 0 for all symbols except `type`. The second problem is also solved through creation of new variables (and variable nodes). In this case, the issue arises because conditions without variables become detached from the graph, and thus do not constrain the instantiations generated. The solution adds a common production variable to all of the conditions in a production, enabling the corresponding variable node to determine whether all of the conditions match.

Binding confusion arises because the graph independently tracks the legal bindings of each variable – called *instantiationless match* in [28] – rather than maintaining Rete's explicit combinations of condition instantiations. Suppose `(A ^type B)`, `(C ^type D)`, `(B ^color Red)` and `(D ^color Blue)` are in working memory. The match binds v_0 to `A` & `C`, v_1 to `B` & `D`, and v_2 to `Red` & `Blue`, but it can't, for example, distinguish which color (v_2) to associate with object `A` (v_0), even though a correct match requires `Red` rather than `Blue`.

This problem is a direct consequence of the interaction between two constraints imposed by factor graphs: (1) the *locality* of processing in the network; and (2) the limiting of message content to the values of one variable. Approaches to binding confusion must either work around these constraints to yield correct combinations or redefine match to live within them. Correct combinations can be yielded, for example, by post-extraction [29] or by implementing Rete. If instead match is to be redefined to be what is produced, we must then determine how to write rules that yield the desired overall behavior given the new semantics. This approach could also

be further refined by replacing the Boolean values in working memory with apportioned fractional values to represent ambiguity in the binding combinations.

The most promising approach at this point modestly redefines the semantics of match to produce the needed combinations of bindings for action variables, while still avoiding creation of Rete's full instantiations. In the process, it eliminates binding confusion, alters the worst-case match cost for a production to exponential in its *treewidth* (see below), and avoids the unnecessary cost and potential confusion engendered when Rete creates redundant instantiations; that is, instantiations differing in bindings of condition variables but not in bindings of action variables. Binding confusion is eliminated by extending variable nodes from their default role of representing individual production variables to representing combinations of production variables (Figure 6). First, an ordering is imposed on the production's conditions and actions to yield a sequence of factor nodes, one for each condition and action. A variable node is then added between each successive pair of factor nodes. Finally, to determine which production variables are assigned to each variable node, the first and last condition or action that uses each production variable is determined, and that variable is added to each variable node between the corresponding factor nodes. The approach is based on *stretching* in factor graphs, which itself maps onto junction trees [16]. Its implementation eliminates binding confusion by tracking combinations of variable bindings just as they are needed. The treewidth in such a structure is simply the maximum number of variables at a node; two in this case.



$$\mathbf{P_1}(V_0,V_1,V_2) = \mathbf{C_1}(V_0,V_1)\mathbf{C_2}(V_1,V_2)\mathbf{A}_1(V_0,V_2)$$

Figure 6. Modified rule graph

Since individual variable nodes in the graph may now represent multiple production variables, messages can be multi-dimensional arrays that are expensive to process without further optimization. The most critical optimization here is the kind of factor rearrangement that is enabled by the distributive law. Without it, the full factor graph for the rule in Figure 4 – comprising 8 factor nodes and 8 variable nodes when all three issue solutions are included along with a *goal memory* added in analogy to working memory in order to explore backward chaining – exhausts heap space before match completes (in LispWorks PE). With factor rearrangement, match takes only 1.7 sec.

A second critical optimization leverages the uniformity of WM and message arrays – they are almost all 0s or 1s – via an N-dimensional generalization of *region quad/octrees* called *exptrees*, which are akin to the CPT-trees used in Bayesian networks [30]. If an array is uniform, it becomes a single-valued unit. Otherwise, each dimension is bisected to yield 2^N sub-arrays, and the process recurs. The sum and product algorithms are trickier here, but have been worked out. With this optimization, match time is reduced by a further factor of ~7 (from 1.7 to .25 sec). Using exptrees enables match of the rule in Figure 4 to complete even without factor rearrangement, but it then still requires 132 sec. This implies that factor rearrangement, at least for this rule with exptrees, speeds up match by a factor of ~500.

One interesting implication of representing WM as an exptree is that it effectively becomes a piece-wise constant function. Extending this to piece-wise linear functions, as is being investigated in follow on work, holds the potential for effectively and uniformly handling the kinds of continuous variables that are necessary for the processing of probabilities and signals.

4 Revisiting Soar's Decision Cycle via Alchemy

Soar's decision cycle, comprising an elaboration phase and a decision, is the lowest level at which knowledge may affect decisions, at which multiple fragments of knowledge may combine, and at which k-search may involve more than one cycle of match and firing. It is also the key scale at which extending Soar beyond symbol processing could lead to radically expanded functionality and at which it first makes sense to consider incorporation of Soar 9's diversity.

Any reimplementation of Soar's elaboration phase must support its three core functions: (1) elaborating the description of the current situation in working memory based on relevant long-term knowledge; (2) generating operator preferences based on this elaborated working memory; and (3) altering working memory to reflect the application of selected operators. The first two functions are mostly monotonic, while the third is inherently non-monotonic. Overall, operation is similar to that of a truth maintenance system [31], with operators determining the current assumptions and elaborations automatically asserting and retracting as these assumptions change.

A second constraint on any reimplementation of the elaboration phase is that it must be executable in bounded time and space. Soar's production-based elaboration phase runs in time that is bounded by the *volume* of the elaboration phase – cost per production × number of productions × number of elaboration cycles. In reality, the second dimension is close to constant, as a suitably optimized Rete algorithm enables match time to remain close to constant with growth in the number of productions [32]. However, the other two dimensions can be problematic. As mentioned earlier, the cost per production may be exponential in the number of conditions in the production. This can be reduced to exponential in the production's treewidth via the approach in the previous section, but this is still combinatoric. Even worse, the length of the elaboration phase can be infinite – new working memory elements can be generated on each elaboration cycle that lead to more productions firing in the next cycle. A reimplementation should at least avoid exacerbating these boundedness issues, and ideally should improve them.

Beyond these two constraints, the uniform versions of Soar also lived with a third constraint that all long-term knowledge must be cast as productions. Productions have the advantage that they are uniform, active, relatively flexible, modular and learnable. They also have a long successful history in cognitive modeling. Still, they do not cope well with declarative and perceptual knowledge, leading to the ultimate elimination of this long held constraint in Soar 9 and the addition of three new long-term memories – two for declarative knowledge (semantic and episodic) and one for perceptual knowledge (imagery) – each with its own distinct form of knowledge. The approach explored here is not to eliminate this third constraint, but to replace it with one based on the varieties of knowledge structures efficiently implementable via graphical models. The hope is thereby to support both hybrid and mixed functionality, as well as Soar 9's new kinds of knowledge structures, in a general yet uniform fashion.

Rather than building on the implementation of production match described in the previous section, a tactical decision was made to explore a hybrid mixed decision cycle via an existing graphical language that already combines some form of symbolic and probabilistic reasoning, such as Alchemy, BLOG or FACTORIE. This was done to help understand the potential applicability of such languages to cognitive architecture, to see if they could provide a short cut to the ultimate goal of simple-yet-diverse architectures, and because of an unresolved issue with the existing implementation that was blocking further progress. For general probability and signal processing, cycles of bidirectional message passing are required across the graph. For example, for sequential signal processing – as is found in the kinds of hidden Markov models used in speech – bidirectional message processing needs to occur across a trellis diagram, in

which a graph is constructed as a linked sequence of identical subgraphs. The existing implementation could use bidirectional message passing within rules in computing the match, but only communicated across rules through working memory. The initial hope had been to combine forward and backward chaining for bidirectional message passing across rules, but this has not so far panned out.

Alchemy, which combines first-order logic and Markov networks in the form of *Markov logic*, was ultimately selected because it: supports forms of both symbolic and probabilistic processing along with nascent signal processing [33]; provides an obvious approach to working with both rules and their instantiations; is publically available, runs on multiple types of computers, and has manuals, tutorials, and rapid response to emailed questions. In Alchemy, a *Markov logic network* (MLN) is defined via first-order predicates and formulas, with weights assigned to the formulas; for example, the Alchemy sentence "`(Color(o1,c)^Type(o2,o1)) => Color(o2,c)`." expresses simple color inheritance, à la Figure 4, while the sentence "`10 Color(F, Green)`" states that the particular object F is green with a weight of 10. The MLN is then compiled into a *ground Markov network* with binary nodes for each ground predicate – such as the fact that F is green – links among nodes that appear in common formulas, such as would be generated between `Color(F,Green)` and `Type(E,F)` by their co-occurrence in at least one grounding of the implication; and features for each possible ground formula. Inference is performed on this ground Markov network, unless additional optimizations such as laziness (where grounding only occurs for variables that take on non-default values [34]) or lifting (where multiple ground atoms are combined into single network nodes when they can be guaranteed to pass the same messages during belief propagation [35]) are included.

The initial mapping of Alchemy to Soar's decision cycle has focused on the first two functions of the elaboration phase: elaborating the current situation in working memory based on the contents of (a rule-based) long-term memory, and generating preferences. Productions are represented as conditional formulas in an MLN file and the state of working memory at the beginning of the decision cycle is represented as evidence in an Alchemy database file. A single elaboration phase is then mapped onto an invocation of Alchemy's inference procedure with this network and database. Given the MLN file and the evidence, Alchemy compiles the productions into a ground Markov network, and then performs inference in this ground network. Because nodes in this network correspond to working memory elements, and each such node links to every other element with which it coexists in a ground formula, the ground Markov network provides a single linked network for the entire elaboration phase, enabling the requisite bidirectional inference across structures. If the productions define a trellis by repetition across elaboration cycles, bidirectional inference also occurs appropriately for it.

Several small-scale experiments have been run to help understand the relevance of this approach to the specific problem of building uniform graphical cognitive architectures and to the general problem of building hybrid mixed decision cycles.[1] These experiments amounted to: (1) re-implementing the simple production systems that were previously implemented via factor graphs; (2) adding a form of semantic long-term memory to the production memory; (3) exploring an implementation of the eight puzzle, one of the earliest tasks investigated in Soar [36] and the basis for early learning experiments with it [23]; and (4) experimenting with trellis diagrams. Several of these experiments have also been replicated with BLOG, but the results are not fundamentally different from those described here based on Alchemy.

[1] Alchemy was also earlier explored in the context of the Icarus cognitive architecture [37], but with a specific focus on the implementation of inference [38].

The details of these experiments are not terribly interesting, so the strategy here is to ignore the nitty-gritty and instead focus on the high level lessons learned from the experiments. The two main lessons are that: (1) a simple mixed decision cycle can be implemented in this fashion, yielding an elaboration phase that combines Soar's standard rule-based capabilities with probabilistic reasoning, simple trellises and semantic memory (fact storage); and (2) there are sufficient issues with it to suggest that it does not provide a short cut to the ultimate goal. The remainder of this section focuses on these issues, and on what can be learned from them for achieving the ultimate goal.

One issue with the Alchemy approach that immediately pops out is that match actually occurs during the compilation of the (first-order) Markov logic network to the ground Markov network, rather than proceeding via the kind of within-network inference that occurred in the factor graph implementation. In essence, the Markov logic network corresponds to the definition of the production system while the ground Markov network corresponds to working memory elements (the ground nodes) and production instantiations (the ground formulas). In contrast to the factor graph implementation, working memory elements are represented by distinct nodes in this network rather than simply serving as the basis for messages among nodes. Given that the ultimate goal is to implement a broadly functional cognitive architecture uniformly in graphs, this match-as-compilation approach is problematic. The resulting inhomogeneity is one of the main reasons why subsequent work has moved back into factor graphs, where the question can be asked as to whether it is possible to unify match – i.e., the computation of ground instances from first-order formulas – with the other necessary forms of probability and signal processing into a single graph that is processed in a uniform manner, or whether it will be necessary to go with something like Alchemy's dual graph/network approach in which match occurs via a first-order graph that generates a ground graph within which the remaining inference occurs.

Despite this fundamental difference in how match is implemented, it turns out that exptrees serve a role in the factor graphs that is analogous to the use of laziness and lifting in Alchemy. The latter mechanisms eliminate unnecessary computation, either by avoiding the processing of default values or by grouping items that can be treated the same. With exptrees, defaults are identified naturally and neighboring items are grouped by region if their values are identical. Exptrees appear to be a coarser approach, but it may ultimately be possible to bring these approaches more into alignment as differences between the approaches are better understood.

The other major issue worth mentioning is a broad one concerning search, times scales, locality of processing, and non-monotonicity. Systems like Alchemy that combine general first order reasoning with probabilistic inference perform global propagation of information in their knowledge structures in service of reaching global minima in a solution space. In other words, they are engaged in search; and, in particular, combinatoric search that can easily get stuck in local minima [38]. This clearly has the essential character of problem space search rather than knowledge search, although it has been mapped here onto Soar's decision cycle. Even though there is no distinction between k-search and ps-search in such systems, this mapping suggests that perhaps should be. Combinatoric search needs to be controlled by knowledge, which itself must be processed in a more tractable fashion. This leads to the hypothesis that Alchemy, and in fact this whole class of systems, might be more effective in solving real problems if they bounded what they tried to compute within a single settling of the graph – essentially only striving for local minima based on global propagation of information – while adding an explicit capability for controlled search over a sequence of such local minimas to find global minima.

Taking this a step further, such an approach enables assigning simple yet principled functional characterizations to the three time scales at which Soar is defined: activity in a problem space (≥ 1 sec) finds global minima; the decision cycle (100 ms) finds local minima based on global information propagation; and the elaboration cycle (10 ms) only provides local propagation of information. Spatially, the 10 ms scale corresponds to neural circuits, so this last choice is plausible from this perspective. Such an assignment of functionality provides a simple yet powerful view of these cognitive scales that could be applicable to any cognitive architecture. Lack of consistency with this pattern may then reveal incoherence in the architectural definition.

This novel mapping of functionality onto architectural levels in Soar does in fact reveal some potential issues. The biggest one is the use of working memory to enable global communication within elaboration cycles. If the mapping is correct, then this is too powerful for this time scale. Global communication instead should be limited to decision cycles. Interestingly, although rules are used in many architectures, this conflict only exists in those like Soar, where rules map onto the 10 ms level. In most architectures rules map onto the 100 ms level, where global communication is fine. In ACT-R, for example, rules are the objects of selection, and execute sequentially. It has been known for some time that ACT-R's rules actually map onto Soar's operators rather than Soar's rules, with Soar's rules mapping onto ACT-R's subsymbolic processing rather than its rules. Yet, prior to this analysis there was no reason to select between these two distinct alternative levels for rules. It is important to note though that this analysis does not ban rules at the 10 ms level, only global rather than local communication among them.

One less momentous aspect of this issue concerns non-monotonic reasoning. Architectures such as Soar embody non-monotonic reasoning in various forms. Operator application, for example, is inherently non-monotonic in the changes it makes to the current state. Soar also maintains an implicit closed world assumption with respect to working memory – that anything not present is false – enabling negated conditions to implement negation as failure, a form of default reasoning. Non-monotonic reasoning is difficult in general in a first order reasoner such as Alchemy. It is also problematic within rules, from a levels perspective, because it is implicitly global. Default reasoning, for example, draws conclusions from a lack of evidence; that is, based on a global assumption about what does not exist. Operator application in Soar does not engender a level conflict because it occurs at the 100 ms level, but negated conditions do because they are defined at the level below.

In contrast to the elaboration phase, there are many fewer constraints on the decision procedure that follows it. Decisions in Soar were based on vote counting in a very early version, on symbolic preferences – acceptable, reject, better worse, etc. – in most versions, and on a combination of symbolic and (additive) numeric preferences in Soar 9. The key constraint on a reimplementation of the decision procedure is that all of the preferences accessed during the elaboration phase must be combined in an appropriate and tractable manner to yield either the selection of a unique operator or the detection of an impasse. Limited experiments with decisions have been performed by leveraging Alchemy's provision of weights on formulas to encode preferences, and most-probable-explanation (MPE) inference to select operators based on these preferences. This has proven adequate for simple examples, but developing a full decision mechanism is left for future work.

5 Conclusion and Next Steps

The ultimate goal for work in cognitive architecture should be architectures that are comprehensive models of human cognition and/or effective architectures for constructing

human-like artificial intelligence. To reach this goal, it is important to resolve the diversity dilemma, enabling the development of architectures that are functional and efficient yet simple, elegant, extensible, integrable and maintainable. This article proposes achieving this by combining architectural diversity with a uniform implementation level based on graphical models. As initial evidence for the plausibility of this approach, experiments have been described that reimplement Soar's lowest two architectural levels – the elaboration and decision cycles – via graphical models. The latter begins the process of investigating a hybrid mixed architecture by exploring simple versions of the kinds of trellises used in sequential signal processing and the use of weights on productions to encode preferences.

Many outstanding issues remain with these partial reimplementations, but they start to reveal the potential of a uniform, graphical implementation level to support diverse cognitive architectures. Current work is focused on completing a uniform implementation of a hybrid mixed decision cycle based on factor graphs that is capable of combining symbol, probability and signal processing to integrate together procedural knowledge, declarative knowledge (e.g., semantic and episodic memories) and perceptual knowledge. Learning from the experiments with Alchemy, and the resulting analysis, this work is paying careful attention to locality of computation. It is, for example, looking at how to keep the elaboration cycle a local rather than a global process, by directly linking actions of some productions with conditions of others, rather than going through a global WM. There is also a major focus on understanding how to perform production match and probabilistic processing compatibly via message passing, rather than relegating the former to a precompilation phase.

Future work will include reimplementing much of the rest of Soar 9 – including semantic and episodic memories, reflection and learning – while enhancing it with additional capabilities, such as decision-theoretic planning, Markov decision processes, and theory of mind. Beyond Soar, it will also be essential to explore: reimplementation of other leading architectures, as well as hybrids among them; new architectures that are more directly inspired by the uniform multipotency of a graphical implementation level; and the potential of graphical models to bridge the gap between symbolic and neural architectures.

Acknowledgments

This article is based on three papers published during 2009, a conference paper that introduced the cognitive hourglass and focused on a graphical reimplementation of the elaboration cycle [39], a workshop paper that explored a mixed elaboration phase via Alchemy [40], and a symposium paper that introduced the diversity dilemma [41]. This body of work was made possible by sabbatical support from the USC Viterbi School of Engineering plus funding from the Institute for Creative Technologies (ICT). ICT's Cognitive Architecture Working Group has been invaluable for semi-public exploration of these ideas. [16] first attracted my attention to factor graphs and is the basis for much of the general material here on them. I would also like to thank the Alchemy group at the University of Washington for their help in installing Alchemy and working through various issues that arose during experimentation with it.

References

[1] P.S. Rosenbloom, J.E. Laird, A. Newell, The Soar Papers: Research on Integrated Intelligence, MIT Press, Cambridge, MA, 1993.
[2] J.R. Anderson, Rules of the Mind, Erlbaum, Hillsdale, NJ, 1993.

[3] J.E. Laird, Extending the Soar cognitive architecture, in Artif. Gen. Intell. 2008: Proc. of the 1st AGI Conf., IOS Press, 2008.
[4] J.E. Laird, P.S. Rosenbloom, The evolution of the Soar cognitive architecture, in D.M. Steier, T.M. Mitchell (Eds.), Mind Matters: A Tribute to Allen Newell, Erlbaum, Mahwah, NJ, 1996.
[5] P.S. Rosenbloom, A. Newell, J.E. Laird, Towards the knowledge level in Soar: The role of the architecture in the use of knowledge, in K. VanLehn (Ed.), Architectures for Intelligence, Erlbaum, 1991.
[6] P.S. Rosenbloom, A cognitive odyssey: From the power law of practice to a general learning mechanism and beyond, Tutor. in Quant. Methods for Psychol. 2 (2006) 43-51.
[7] D.J. Pearson, N.A. Gorski, R.L. Lewis, J.E. Laird, Storm: A framework for biologically-inspired cognitive architecture research, in Proc. of the 8th Intl. Conf. on Cogn. Model., 2007.
[8] R. Cooper, J. Fox, COGENT: A visual design environment for cognitive modeling, Behav. Res. Methods, Instrum. & Comput. 30 (1998) 553-564.
[9] A. Newell, Unified Theories of Cognition, Harvard Press, Cambridge, MA, 1990.
[10] S. Deering, Watching the waist of the protocol hourglass, Keynote at ICNP '98, 1998.
[11] P. Domingos, What is missing in AI: The interface layer, in P. Cohen (Ed.), Artificial Intelligence: The First Hundred Years, AAAI Press, Menlo Park, CA, In press.
[12] B. Cho, P.S. Rosenbloom, C.P. Dolan, Neuro-Soar: A neural-network architecture for goal-oriented behavior, in Proc. of the 13th Annual Conf. of the Cogn. Sci. Soc., Erlbaum, 1991, pp. 673-677.
[13] D.J. Jilk, C. Lebiere, R.C. O'Reilly, J.R. Anderson, SAL: An explicitly pluralistic cognitive architecture, in J. of Exp. and Theor. Artif. Intell. 20 (2008) 197-218.
[14] M.I. Jordan, Graphical models, in Stat. Sci. 19 (2004) 140-155.
[15] J. Pearl, Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference, Morgan Kaufman, San Mateo, CA, 1988.
[16] F.R. Kschischang, B.J. Frey, H. Loeliger, Factor graphs and the sum-product algorithm, in IEEE Trans. on Inf. Theory 47 (2001) 498-519.
[17] M.I. Jordan, T.J. Sejnowski, Graphical Models: Foundations of Neural Computation. MIT Press, Cambridge, MA, 2001.
[18] V. Gogate, R. Dechter, Approximate Inference Algorithms for Hybrid Bayesian Networks with Discrete Constraints, in Proc. of the 21st Conf. on Uncertain. in Artif. Intell., 2005, pp. 209-216.
[19] A. McCallum, K. Rohanemanesh, M. Wick, K. Schultz, S. Singh, FACTORIE: Efficient probabilistic programming via imperative declarations of structure, inference and learning, in Proc. of the NIPS Workshop on Probab. Program., 2008.
[20] B. Milch, B. Marthi, S. Russell, D. Sontag, D.L. Ong, A. Kolobov, BLOG: Probabilistic models with unknown objects, in L. Getoor, B. Taskar (Eds.), Introduction to Statistical Relational Learning, MIT Press, Cambridge, MA, 2007.
[21] P. Domingos, S. Kok, H. Poon, M. Richardson, P. Singla, Unifying logical and statistical AI, in Proc. of the 21st Natl. Conf. on Artif. Intell., AAAI Press, 2006, pp. 2-7.
[22] C.L. Forgy, Rete: A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem, in Artif. Intell. 19 (1982) 17-37.
[23] J.E. Laird, P.S. Rosenbloom, A. Newell, Chunking in Soar: The anatomy of a general learning mechanism, in Mach. Learn. 1 (1986) 11-46.
[24] W. Schneider, R.M. Shiffrin, Controlled and automatic human information processing: I. Detection, search, and attention, Psychol. Rev. 84 (1977) 1-66.

[25] M. Mézard, G. Parisi, R. Zecchina, Analytic and algorithmic solution of random satisfiability problems, Sci. 297 (2002) 812-815.
[26] K.A. Bonawitz, Composable Probabilistic Inference with Blaise, Doctoral Dissertation, Department of EECS, MIT, Cambridge, MA, 2008.
[27] J.S. Yedidia, W.T. Freeman, Y. Weiss, Constructing free-energy approximations and generalized belief propagation algorithms, IEEE Trans. on Inf. Theory 51 (2005) 2282-2312.
[28] M. Tambe, P.S. Rosenbloom, Investigating production system representations for non-combinatorial match, Artif. Intell. 68 (1994) 155-199.
[29] R. Dechter, J. Pearl, Network-based heuristics for constraint-satisfaction problems, Artif. Intell. 34 (1987) 1-38.
[30] C. Boutilier, N. Friedman, M. Goldszmidt, D. Koller, Context-specific independence in Bayesian networks, in Proc. of the 12th Conf. on Uncertain. in Artif. Intell., Morgan Kaufman, 1996, pp. 115-123.
[31] J. Doyle, A truth maintenance system, Artif. Intell. 12 (1979) 251-272.
[32] R.B. Doorenbos, Matching 100,000 learned rules, in Proc. of the 11th Natl. Conf. on Artif. Intell., 1993, pp. 290-296.
[33] J. Wang, P. Domingos, Hybrid Markov logic networks, in Proc. of the 23rd AAAI Conf. on Artif. Intell., AAAI Press, Menlo Park, CA, 2008, pp. 1106-1111.
[34] H. Poon, P. Domingos, M. Sumner, A general method for reducing the complexity of relational inference and its application to MCMC, in Proc. of the 23rd AAAI Conf. on Artif. Intell., AAAI Press , Menlo Park, CA, 2008, pp. 1075-1080.
[35] P. Singla, P. Domingos, Lifted first-order belief propagation, in Proc. of the 23rd AAAI Conf. on Artif. Intell., AAAI Press, Menlo Park, CA, 2008, pp. 1094-1099.
[36] J.E. Laird, A. Newell, A universal weak method: Summary of results, in Proc. of the 8th Intl. Joint Conf. on Artif. Intell., William Kaufmann, 1983, pp. 771-773.
[37] P. Langley, D. Choi, A unified cognitive architecture for physical systems, in Proc.of the 21st Natl. Conf. on Artif. Intell., AAAI Press, Menlo Park, CA, 2006, pp. 1469-1474.
[38] D. Stracuzzi, Personal communication, 2009.
[39] P.S. Rosenbloom, Towards a new cognitive hourglass: Uniform implementation of cognitive architecture via factor graphs, in Proc. of the 9th Intl. Conf. on Cogn. Model., 2009.
[40] P.S. Rosenbloom, A graphical rethinking of the cognitive inner loop, in Proc. of the IJCAI Intl. Workshop on Graph Struct. for Knowl. Represent. and Reason., 2009.
[41] P.S. Rosenbloom, Towards uniform implementation of architectural diversity, in Proc. of the AAAI Fall Symp. on Multi-Represent. Archit. for Hum.-Lev. Intell., 2009.