Towards a New Cognitive Hourglass: Uniform Implementation of Cognitive Architecture via Factor Graphs

Paul S. Rosenbloom (Rosenbloom@USC.Edu)
Department of Computer Science and Institute for Creative Technologies, 13274 Fiji Way
Marina del Rey, CA 90292 USA

Abstract

As cognitive architectures become ever more ambitious in the range of phenomena they are to assist in producing and modeling, there is increasing pressure for diversity in the mechanisms they embody. Yet uniformity remains critical for both elegance and extensibility. Here, the search for uniformity is continued, but shifted downwards in the cognitive hierarchy to the *implementation level*. Factor graphs are explored as a promising core, with initial steps towards a reimplementation of Soar. The ultimate aim is a uniform implementation level for cognitive architectures affording both heightened elegance and expanded coverage.

Keywords: Cognitive architecture; implementation level; factor graphs; graphical models; production match; Soar

From Architecture to Implementation

A *cognitive architecture* is a hypothesis about the fixed structures underlying thought in active intelligent beings, whether natural or artificial. It consists of a set of interacting mechanisms that can combine with appropriate knowledge to model human intelligent behavior and/or generate artificial intelligent behavior. In the large, a cognitive architecture is a theory about one or more *systems levels* comprising an intelligent being. Newell (1990) discussed a hierarchy of levels (organelles, neurons, neural circuits, deliberate acts, operations, etc.) across four bands of human action: biological, cognitive, rational, and social. At each level, a combination of structures and processes implements the basic elements at the next higher level.

One controversial attribute of systems levels in cognitive architecture is their *girth*; i.e., their uniformity versus diversity. 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. Uniformity appeals to simplicity and elegance. In caricature, it is the physicist's approach, where a broad diversity of phenomena emerges from interactions among a small set of general elements. Diversity appeals to specialization and optimization. It is the biologist's approach, in which many specialized structures, each locally optimized, jointly yield a robust and coherent whole.

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 (Deering, 1998). 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*. Domingos's (In press) recent call for an *interface layer* in AI is an appeal for a similar sort of mesoscale uniformity in AI.

Intelligence clearly entails diversity in the cognitive hierarchy across levels, but what about within levels? At the top, the extraordinary range of possible behaviors and applications 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 (with adjustments in Newell's lower levels for grounding in such technologies). But is there an hourglass or a rectangle in between?

The question of the existence of a *cognitive hourglass* has traditionally been cast in terms of whether the cognitive architecture is uniform. Among architectures for cognitive modeling, Soar (Rosenbloom, Laird & Newell, 1993) has been a standard exemplar of uniformity and ACT-R (Anderson, 1993) of diversity. Recently, based on both functional and modeling considerations, Soar 9 (Laird, 2008) has shifted strongly towards diversity, and is helping to tip the community balance in this direction.

As a scientist, one can respond to a demonstrated need for diversity by simply accepting it, or by hypothesizing an underlying uniformity and simplicity that explains it. Anderson, for example, developed a background theory of cognitive rationality to justify ACT-R's mechanisms as optimal adaptations to the environment (Anderson, 1990). The theory's uniformity is not in the architecture itself, but does yield a simple, well-motivated explanation for it. Yet something significant is lost when the uniformity is not in the cognitive hierarchy, as diversity negatively impacts both the elegance of the resulting system and the ease with which new capabilities can be integrated into a unified whole. Historically, diverse architectures have been tough to unify. To the extent such a system remains disunified, it is more of a toolkit or language than a hypothesis about the fixed structures of thought (i.e., an architecture).

Another alternative is not simply to accept diversity, or try to justify it, but to continue a search for uniformity – the narrow waist of the hourglass – elsewhere in the cognitive hierarchy. This is an application of the *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, it 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.

The history of Soar well illustrates the uniformity-first strategy (Laird & Rosenbloom, 1996). For years it had a single procedural, rule-based, long-term memory and a single learning mechanism (Laird, Rosenbloom & Newell, 1986), while investigations continued into their ability to support a diversity of memory (e.g., procedural, semantic, and episodic (Rosenbloom, Newell & Laird, 1991)) and learning (e.g., skill and knowledge acquisition, generalization and transfer, and learning from observation (Rosenbloom, 2006)) 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) (Laird, 2008).

Uniformity-first, however, entails that acknowledging a need for diversity at the architecture level should be accompanied by a continued search for uniformity at other levels. In this article, the particular focus is on burrowing beneath the diversity at the architecture level to look for uniformity at the *implementation level* just below. The goal is still an hourglass, albeit one with a lower waistline.

The implementation of cognitive architectures, while critical for efficiency and usability, is usually extra-theoretic and not part of the architectural hypothesis. Characteristic examples include the COGENT (Cooper & Fox, 1998) and Storm (Pearson, Gorski, Lewis & Laird, 2007) environments for cognitive modeling/architectures, both of which are coarse-grained, graphical tools intended to assist the developer rather than theoretical hypotheses about the implementation level. The primary exception is systems like SAL (Jilk, Lebiere, O'Reilly & Anderson, 2008) or Neuro-Soar (Cho, Dolan & Rosenbloom, 1991), where a cognitive architecture is implemented via neural networks. Neural approaches remain interesting possibilities for the implementation level, but the focus here is on the related but more general class of *graphical models* (Jordan, 2004).

Graphical models share the core graphical/network nature of neural networks and graphical modeling environments, but focus on representing independence across variables in complex functions such as joint probability distributions and communication codes. They include Bayesian networks (Pearl, 1988) and Markov networks, with origins in probabilistic reasoning. But they also include factor graphs (Kschischang, Frey & Loeliger, 2001), which take a broader multivariate-function view. Interestingly, a variety of neural network algorithms – such as supervised Boltzmann machines, radial basis functions, and unsupervised learning algorithms – can be mapped onto graphical models (Jordan & Sejnowski, 2001). A core premise of this article is that graphical models provide untapped potential for cognitive architectures. They may also ultimately forge a principled bridge between neural and symbolic architectures.

The work in this article is based on factor graphs. Although originating in coding theory, where they underlie the "astonishing performance" of turbo codes, factor graphs are particularly promising for cognitive architecture because of the diversity of important problems and algorithms they subsume in a uniform manner when combined with their canonical sum-product algorithm. Factor graphs are relevant for *signal processing*, where they are useful in vision (Drost & Singer, 2003) and subsume Kalman filters, the Viterbi algorithm, and the forward-backward algorithm in hidden Markov models; *probabilistic processing*, where they subsume belief propagation in Bayesian and Markov networks; and *symbol processing*, where they yield arc consistency for constraint problems (Dechter & Mateescu, 2003). There is also significant work on *mixed* approaches combining symbolic and probabilistic processing, *hybrid* approaches combining discrete and continuous processing, and *hybrid mixed* approaches (Gogate & Dechter, 2005).

Factor graphs raise the possibility of a uniform implementation level that elegantly explains 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, breaking down the barriers between central and peripheral processing by bringing the latter within the cognitive inner loop and making each 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. They 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.

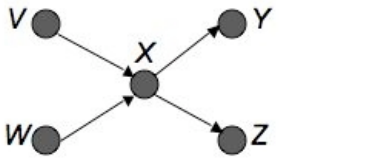
The remainder of this article introduces factor graphs, begins exploring their utility for cognitive architectures via a first step towards a graphical reimplementation of Soar, and summarizes the path forward from here. The focus is not on a specific cognitive model, but on the possibility of a radically new generation of integrated cognitive models.

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, $\mathbf{C}(A,B,C,D)$, 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* (Kschischang, Frey & Loeliger, 2001) – $\mathbf{K}(S_0,I_0,O_0,N_0,\ldots,S_n,I_n,O_n,N_n)$.

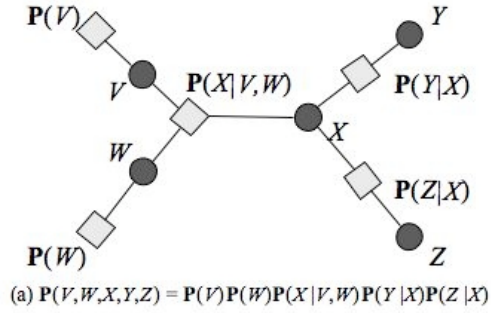
The prototypical factor graph operation is the computation of *marginals* on variables. For a joint probability distribution, this is simply the marginal of a random variable, computed by summing out the other variables: $\mathbf{P}(Y) = \Sigma_{v,w,x,z}\mathbf{P}(v,w,x,Y,z)$. The key to tractability 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: $\mathbf{P}(V,W,X,Y,Z) = \mathbf{P}(V)\mathbf{P}(W)\mathbf{P}(X|V,W)\mathbf{P}(Y|X)\mathbf{P}(Z|X)$. Such decompositions derive from the *chain rule* plus *conditional independence* assumptions. Using commutative and distributive laws then enables more efficient marginal computation: $\mathbf{P}(Y) = \Sigma_x\mathbf{P}(Y|x)\Sigma_z\mathbf{P}(z|x)\Sigma_v\mathbf{P}(v)\Sigma_w\mathbf{P}(x|v,w)\mathbf{P}(w)$. This provides the basics of Bayesian networks (Figure 1).



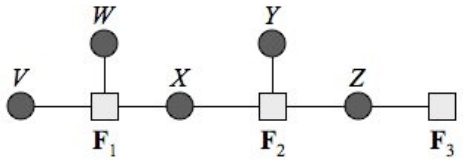
$\mathbf{P}(V,W,X,Y,Z) = \mathbf{P}(V)\mathbf{P}(W)\mathbf{P}(X|V,W)\mathbf{P}(Y|X)\mathbf{P}(Z|X)$

Figure 1. Example Bayesian network

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 function use, and undirected links between factors and their variables (Figure 2).



(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 2. Example factor graphs

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 summarizes the source node's information about the domain of the link's variable node. A message from a variable node to a factor node is the pointwise product of the messages into the variable from all of its neighbors except the target node. A message from a factor node to a variable node starts with this same product but also includes the factor node's own function in the product, with all variables other than the target variable then being summed out to form the 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 graphs in which only a single marginal is desired, the factor graph can be reduced to an *expression tree* in which the products and sums are computed unidirectionally upwards in the tree. Beyond this simplest case, the algorithm works iteratively by sending output messages from nodes as they receive input messages. For *polytrees*, which have at most one undirected 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 uses 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 other pairs.

To improve the efficiency of the algorithm, various optimizations can be applied, and alternative algorithms can be used (such as survey propagation (Mézard, Parisi & Zecchina, 2002) and Monte Carlo sampling (Bonawitz, 2008)). A connection exists between factor graphs and statistical mechanics, revealing that the sum-product algorithm minimizes the *Bethe free energy*, and yielding algorithmic innovations (Yedidia, Freeman & Weiss, 2005).

Factor Graphs for Cognitive Architecture

The key question for us is whether factor graphs can yield a uniform level for implementing, understanding and exploring cognitive architecture, while ultimately yielding novel architectures that are more uniform, unified, and functional. Existing work on hybrid mixed methods is encouraging, as is work on general languages for mixed probabilistic and logical reasoning. FACTORIE (McCallum, Rohanemanesh, Wick, Schultz & Singh, 2008), for example, combines factor graphs with an imperative programming language to support relations and other capabilities, while BLOG (Milch, Marthi, Russell, Sontag, Ong & Kolobov, 2007) and Alchemy (Domingos, Kok, Poon, Richardson & Singla, 2006) 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 factor graphs, existing architectures, and the implications of implementing architectures via factor graphs; (2) go beyond

existing architectures by hybridization and simplification, both within and across architectures; and (3) integrate in new capabilities that don't mesh well with existing architectures, such as perception and motor control.

The initial focus is on Soar because of familiarity with it, its history in cognitive modeling, and its dual status as both a uniform (early) and a diverse (latest) architecture. We can make a quick start on reimplementing the uniform core, and build towards a more uniform integration of later diversity.

The inmost core of "uniform Soar" is the *reactive layer*, where working memory (WM) is elaborated via associative retrieval of relevant information from a parallel production system. During a single *elaboration cycle*, match computes all legal production instantiations, which then fire in parallel to modify WM. Match is the core of the elaboration cycle, so it is the natural focus for initial reimplementation efforts.

In Soar, match is based on *Rete* (Forgy, 1982), comprising a discrimination network for sorting working memory elements (wmes) to production conditions, a join network to determine which combinations of wmes yield production instantiations (while attending to across-condition variable equality), 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.

A factor graph implementation of Rete has been designed, where factor nodes handle discriminations and joins, variable nodes represent wmes that match production conditions and their combinations – analogous to Rete's α and β *memories* – and unidirectional message passing over an expression tree enables incremental and shared match. But, rather than imposing Rete on factor graphs, the primary focus here has been on algorithms that arise more naturally from viewing production match as a multivariate function.

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

Figure 3. Example rule

Consider the rule in Figure 3. This is not exactly Soar's representation, although it does retain its object-attribute-value scheme, with conditions testing wmes via constants and variables (in angle brackets). The simplest 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: $\mathbf{P_1}(v_0,v_1,v_2) = \mathbf{C_0}(v_0,v_1)\mathbf{C_2}(v_1,v_2)$ (Figure 4).

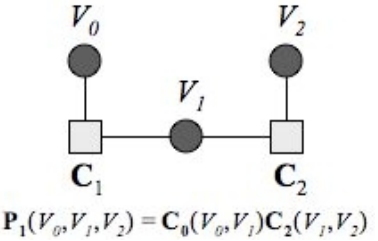


Figure 4. Example rule graph

This mapping has been implemented. In it, 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.

This initial approach showed the feasibility of implementing match via factor graphs, but it also raised three issues: (1) both WM and tests of constants were hidden within the condition factors; (2) production match ignored conditions without variables; and (3) it led to errors from *binding confusion* (Tambe & Rosenbloom, 1994). Solutions for these issues have been implemented, but as the first one didn't affect correctness – only how much factor graphs were leveraged – and the second couldn't actually occur in Soar because all of its conditions must be linked via variables, only the third issue is discussed here.

Binding confusion arose because the graph independently tracked the legal bindings of each variable – called *instantiationless match* in (Tambe & Rosenbloom, 1994) – 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 bound v_0 to A & C, v_1 to B & D, and v_2 to Red & Blue, but it couldn't, for example, distinguish which color (v_2) to associate with object A (v_0), despite the fact that a correct match would mandate Red rather than Blue.

This problem is a direct consequence of the interaction between two types of constraint imposed by factor graphs: (1) the *locality* of processing in the network; and (2) the limitation on 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 (Dechter & Pearl, 1987) 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 Boolean array values with apportioned fractional values for ambiguous bindings.

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 the need for 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*, and further reduces costs and potential confusion by eliminating redundant instantiations that would otherwise generate equivalent results (when some condition-variable bindings differ while action-variable bindings do not).

This approach enables local processing of variable combinations by using variable nodes in the graph that represent combinations of production variables rather than individual ones. To start, an ordering is imposed on the production's conditions and actions to yield a sequence of factor nodes. A variable node is then added between each

successive pair of factor nodes. To finish, the first and last condition/action that uses each production variable is determined, and that variable is added to each variable node between these two factor nodes (Figure 5). The approach is based on *stretching* in factor graphs, which itself maps onto junction trees (Kschischang, Frey & Loeliger, 2001). Its implementation eliminates binding confusion by tracking combinations of variable bindings just as they are needed.

Since each variable node in the graph may now represent multiple production variables, multi-dimensional arrays result that can be expensive to process without further optimization. The most critical optimization here is factor rearrangement. Without it, the full factor graph for the rule in Figure 3 – comprising 8 factor nodes and 8 variable nodes when all three problem solutions are included plus the *goal memory* to be described later – exhausts heap space before match completes (in LispWorks PE). With factor rearrangement, match takes only 1.7 sec.

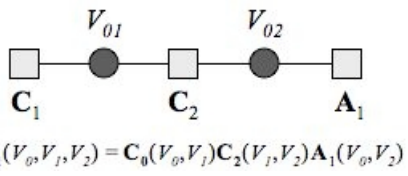


Figure 5. Modified rule graph

A second critical optimization leverages the uniformity of WM and message arrays (which are almost all 0s or 1s) via an N-dimensional generalization of *region quad/octrees* (à la CPT-trees in Bayesian networks (Boutilier, Friedman, Goldszmidt & Koller, 1996)). If an array is uniform, it becomes a single-valued unit. Otherwise, each dimension is bisected – yielding 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.). It also enables comparing match cost without rearrangement, yielding a factor of ~500 (132 vs. .25 sec.).

One interesting implication of representing WM via trees is a view of it as a piece-wise constant function. If this proves extensible to piece-wise linear functions, it may be effective for variables with continuous domains and ranges (as used in mixed and hybrid systems). It may also be possible to employ more intelligent partitioning algorithms for WM, including adaptive clustering methods.

Conclusion and Next Steps

Despite the increasing trend towards diversity in cognitive architectures, uniformity at the implementation level may yet provide leverage in exploring, understanding and improving existing architectures; and in developing novel architectures with increased elegance and broader coverage. Factor graphs, and graphical models more generally, are intriguing for this level because they yield a wide diversity of capabilities in a uniform and constrained manner.

An initial step has been taken towards reimplementing Soar by factor graphs, with the demonstration of the latter's ability to implement (symbolic) production match via an interesting new algorithm. The key next step is extending beyond match to the rest of Soar's cognitive inner loop – the *deliberate layer* (or *decision cycle*) – where elaboration cycles repeat until quiescence (the *elaboration phase*) followed by a decision. One approach to the elaboration phase is to alter WM between cycles, as in standard production systems. This has been implemented, but a more promising alternative is to arrange the elaboration phase's temporal structure in space rather than time, as a trellis. With a trellis, perceptual and motor processing may be integrated directly into the cognitive inner loop rather than being walled off into a separate I/O system. A trellis would also enable bidirectional information propagation across the elaboration phase to ensure correct graphical probability calculations. For the process of decision making itself, *influence diagrams* are a natural strategy to explore first.

Beyond reimplementing Soar's cognitive inner loop is the challenge of extending the loop to include a more uniform integration of Soar 9's semantic and episodic memories, plus probability and signal processing. The lead candidate for semantic memory blends Prolog's view of facts as condition-less rules that are triggered backwards by a goal probe, with the statistical view of retrieving the most probable semantic memory element given the probe (Anderson, 1990). A goal memory – in analogy to working memory – has been implemented to enable backwards access to production actions; but appropriate control of backwards vs. forward processing in the inner loop is still needed, as is restricting retrieval to the most probable element (based on MAP probabilities and the max-product algorithm). For episodic memory, two approaches have potential: (1) adding long-term trellises to the graph; and (2) extending WM to a fourth, temporal dimension.

Adding probabilities to the inner loop is being explored via experiments with extant mixed languages, such as Alchemy and BLOG (Rosenbloom, 2009). Signal processing will be investigated via trellises and piecewise-linear quad/octrees (for representing continuous functions).

Still, this is all only the beginning. It will also be critical to: (1) reimplement Soar's *reflective layer* and learning mechanism(s); (2) implement and integrate in other cognitive capabilities, such as planning, emotion, social cognition and language; (3) reexamine the implementation of a broader range of architectures (such as ACT-R); and (4) forge a bridge to neural architectures. Success should yield both a uniform implementation level for architecture development – i.e., a narrow waist for the cognitive hourglass – and better architectures for cognitive modeling.

Acknowledgments

This effort 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. (Kschischang *et al*., 2001) first attracted my attention to factor graphs and is the basis for much of the general material here on them.

References

Anderson, J. R. (1990). *The Adaptive Character of Thought*. Hillsdale, NJ: Erlbaum.

Anderson, J. R. (1993). *Rules of the Mind.* Erlbaum.

Bonawitz., K. A. (2008) *Composable Probabilistic Inference with Blaise*. Doctoral Dissertation, Department of EECS, MIT, Cambridge, MA.

Boutilier, C., Friedman, N., Goldszmidt, M. & Koller, D. (1996). Context-specific independence in Bayesian networks. *Proceedings of the Twelfth Conference on Uncertainty in AI* (pp. 115-123). Morgan Kaufman.

Cho, B., Rosenbloom, P. S. & Dolan, C. P. (1991). Neuro-Soar: A neural-network architecture for goal-oriented behavior. *Proceedings of the 13th Annual Conference of the Cognitive Science Society* (pp. 673-677). Erlbaum.

Cooper, R. & Fox, J. (1998). COGENT: A visual design environment for cognitive modeling. *Behavior Research Methods, Instruments & Computers*, *30*, 553-564.

Dechter, R. & Mateescu, R. (2003). A simple insight into iterative belief propagation's success. P*roceedings of The Nineteenth Conference on Uncertainty in Artificial Intelligenc*e (pp. 175-183). Morgan Kaufman.

Dechter, R. & Pearl, J. (1987). Network-based heuristics for constraint-satisfaction problems. *Artificial Intelligence*, *34*, 1-38.

Deering, S. (1988). *Watching the waist of the protocol hourglass*, Keynote address at ICNP '98.

Domingos, P. (In press). What is missing in AI: The interface layer. In P. Cohen (Ed.), *Artificial Intelligence: The First Hundred Years*. Menlo Park, CA: AAAI Press.

Domingos, P., Kok, S., Poon, H., Richardson, M. & Singla, P. (2006). Unifying logical and statistical AI. *Proceedings of the Twenty-First National Conference on Artificial Intelligence* (pp. 2-7). AAAI Press.

Drost, R. J. & Singer. A. W. (2003). Image segmentation using factor graphs. *Proceedings of the 2003 IEEE Workshop on Statistical Signal Processing* (pp. 150-153).

Forgy, C. L. (1982). Rete: A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem. *Artificial Intelligence*, *19*, 17-37.

Gogate, V. & Dechter, R. (2005). Approximate Inference Algorithms for Hybrid Bayesian Networks with Discrete Constraints. *Proceedings of the 21st Conference on Uncertainty in Artificial Intelligence* (pp. 209-216).

Jilk, D. J., Lebiere, C., O'Reilly, R. C. & Anderson, J. R. (2008). SAL: An explicitly pluralistic cognitive architecture. *Journal of Experimental and Theoretical Artificial Intelligence*, *20*, 197-218.

Jordan, M. I. (2004). Graphical models. *Statistical Science (Special Issue on Bayesian Statistics)*, *19*, 140-155.

Jordan, M. I. & Sejnowski, T. J. (2001). *Graphical Models: Foundations of Neural Computation*. MIT Press.

Kschischang, F. R., Frey, B. J. & Loeliger, H. (2001). Factor graphs and the sum-product algorithm. *IEEE Transactions on Information Theory*, *47*, 498-519.

Laird, J. E. (2008). Extending the Soar cognitive architecture. In *Artificial General Intelligence 2008: Proceedings of the First AGI Conference*. IOS Press.

Laird, J. E. & Rosenbloom, P. S. (1986). Chunking in Soar: The anatomy of a general learning mechanism. *Machine Learning, 1*, 11-46.

Laird, J. E. & Rosenbloom, P. S. (1996). The evolution of the Soar cognitive architecture. In D. M. Steier. and T. M. Mitchell (Eds.), *Mind Matters: A Tribute to Allen Newell*. Mahwah, NJ: Erlbaum.

McCallum, A., Rohanemanesh, K., Wick, M., Schultz, K. & Singh, S. (2008). FACTORIE: Efficient probabilistic programming via imperative declarations of structure, inference and learning. *Proceedings of the NIPS workshop on Probabilistic Programming*.

Mézard, M., Parisi, G. & Zecchina, R. (2002). Analytic and algorithmic solution of random satisfiability problems. *Science*, *297*, 812-815.

Milch, B., Marthi, B., Russell, S., Sontag, D., Ong, D. L. & Kolobov, A. (2007). BLOG: Probabilistic models with unknown objects. In L. Getoor and B. Taskar, (Eds.) *Introduction to Statistical Relational Learning*. Cambridge, MA: MIT Press.

Newell, A. (1990). *Unified Theories of Cognition*. Cambridge, MA: Harvard University Press.

Pearl, J. (1988). *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. San Mateo, CA: Morgan Kaufman,

Pearson, D. J., Gorski, N. A., Lewis, R. L. & Laird, J. E. (2007). Storm: A framework for biologically-inspired cognitive architecture research. In *Proceedings of the 8th International Conference on Cognitive Modeling*.

Rosenbloom, P. S. (2006). A cognitive odyssey: From the power law of practice to a general learning mechanism and beyond. *Tutorials in Quantitative Methods for Psychology*, *2*, 43-51.

Rosenbloom, P. S. (2009). A graphical rethinking of the cognitive inner loop. In *Proceedings of the IJCAI International Workshop on Graph Structures for Knowledge Representation and Reasoning.*

Rosenbloom, P. S., Laird, J. E. & Newell, A. (1993). *The Soar Papers: Research on Integrated Intelligence*. Cambridge, MA: MIT Press.

Rosenbloom, P. S., Newell, A. & Laird, J. E. (1991). Towards the knowledge level in Soar: The role of the architecture in the use of knowledge. In K. VanLehn (Ed.), *Architectures for Intelligence*. Erlbaum.

Tambe, M. & Rosenbloom, P. S. (1994). Investigating production system representations for non-combinatorial match. *Artificial Intelligence*, *68*, 155-199.

Yedidia, J. S., Freeman, W. T. & Weiss, Y. (2005). Constructing free-energy approximations and generalized belief propagation algorithms. *IEEE Transactions on Information Theory*, *51*, 2282-2312.