Towards a New Cognitive Hourglass

Uniform Implementation of Cognitive Architecture via Factor Graphs

Paul S. Rosenbloom
Department of Computer Science
University of Southern California
rosenbloom@usc.edu

Abstract

As cognitive architectures become ever more ambitious in the scope of intelligent behaviors they are to produce, 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. 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 function.

1 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, when combined with domain knowledge, conjointly yield the capabilities – such as perception, reasoning, motivation, planning, reactivity, learning, and action – required for adaptive behavior in the world. 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 a pastiche of specialized structures, each locally optimized, combine to yield a robust and coherent whole.

Across a hierarchy of levels, there is no a priori reason to assume they are 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 to provide "a language of operations that is all the infrastructure needs to support, and all that the applications need to know about" is an appeal for just this sort of mesoscale uniformity in AI.

Intelligence clearly entails diversity in the cognitive hierarchy. At the top, the extraordinary range of possible applications/behaviors is one of the core phenomena that cognitive architectures are developed to explain. Diversity in implementations is a bit more complex. According to *physicalism*, the mind is grounded in the diverse biology of the brain, and according to *strong AI*, it can also be grounded in a diversity of alternative technologies. But what happens in between? Is there an hourglass or a rectangle?

Traditionally, this question – of the existence of a *cognitive hourglass* – has been cast in terms of whether the cognitive architecture is uniform. Among architectures with pretension to cognitive modeling, Soar [Rosenbloom *et al*., 1993] has been a standard exemplar of uniformity and ACT-R [Anderson, 1993] of diversity (although Society of Mind may be the extreme [Minsky, 1985]). 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 diversity by simply accepting it, and even reveling in it [Jilk *et al*., 2008], or by hypothesizing an underlying uniformity and simplicity that explains it. Anderson developed a background theory of cognitive rationality to justify ACT-R's mechanisms as optimal adaptations to the environment [Anderson, 1990]. The uniformity of this theory is not in the architecture itself, but it does provide a simple, well-motivated explanation for diversity; and might ultimately become incorporated if architectural evolution is itself capable of uniform modeling.

Yet something significant is lost with either of these two specific responses, 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 – whether ACT, Soar 9, or Society of Mind – have been tough to unify. To the extent such a system remains disunified, it is more of a toolkit or language for developing intelligence than a hypothesis about the fixed structures of thought (i.e., an architecture).

Another alternative is not simply to accept diversity, or explain it externally, 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 beneath the diversity, and may lead down an irrevocable path of complexity.

The history of Soar well illustrates the uniformity-first strategy [Laird and Rosenbloom, 1996]. For years it had a single procedural, rule-based, long-term memory and a single learning mechanism [Laird *et al*., 1986], while investigations continued into their ability to support a diversity of memory (e.g., procedural, semantic, and episodic [Rosenbloom *et al*., 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, developed over years of experimentation, inspired the development of Soar 9, a diverse – yet not completely unified – system 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, demands that acceptance of a need for diversity at the architectural level be accompanied by a continued search for uniformity at other levels; that is, continued burrowing beneath this architectural diversity until either a uniform core is found or the evidence for diversity throughout the hierarchy becomes overwhelming. In this article, the particular focus is on shifting the search to the *implementation level*, beneath the architecture. The goal is still an hourglass, albeit one with a lower waistline.

The implementation of cognitive architectures, although critical for efficiency and usability, is generally considered extra-theoretic and not part of the explicit architectural hypothesis. The principal exception is systems like SAL [Jilk *et al*., 2008], where a neural architecture (Leabra [O'Reilly and Munakata, 2000]) implements a cognitive architecture (ACT-R). Neural approaches to implementation remain interesting, particularly when the focus is on human cognition, but there are other as-yet unexplored possibilities that ultimately may prove more promising, or which may yield critical bridges between neural and cognitive architectures.

One particularly intriguing prospect is to base the implementation level on *factor graph*s [Kschischang *et al*., 2001]. Although factor graphs originated in coding theory, where they underlie the "astonishing performance" of turbo codes, they 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(mary)-product algorithm*. Prior work has shown the relevance of factor graphs to *signal processing*, where they can be effective in vision [Drost and Singer, 2003] and subsume Kalman filters, the Viterbi algorithm, and the forward-backward algorithm in hidden Markov models; *probabilistic processing*, where they subsume both Bayesian and Markov networks; and *symbol processing*, where they yield arc consistency for constraint problems [Dechter and Mateescu, 2003]. There is also significant work on *mixed* approaches – combining symbolic and probabilistic processing – and *hybrid* approaches – combining discrete and continuous processing – as well as *hybrid mixed* approaches [Gogate and 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; fusing symbolic and probabilistic reasoning; and providing the conceptual bridge needed in unifying cognitive architecture with neural networks (although not the focus in this article).

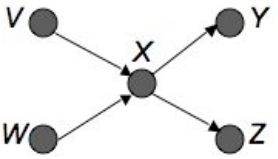
2 Factor Graphs

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, $\mathbf{C}(A,B,C,D)$, 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 *et al*., 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 doesn't involve the direct evaluation of the function, but the computation of the function's *marginal* on one, or all, of the variables. For a joint probability distribution, this is simply the marginal probability of a random variable, as computed by summing out over the values of all of the other variables: $\mathbf{P}(Y) = \Sigma_{v,w,x,z}\mathbf{P}(v,w,x,Y,z)$. The key to tractability is avoiding, in the process, explicit examination of every element of the cross product of the four variables' domains. In probabilistic reasoning, the solution is to decompose the joint distribution into the product of conditional (and prior) probabilities defined on subsets of the 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 are derivable from the *chain rule* along with specific *condi-*

tional independence assumptions. Using commutative and distributive laws then yields rearrangements that enable 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* [Pearl, 1988] (Figure 1).

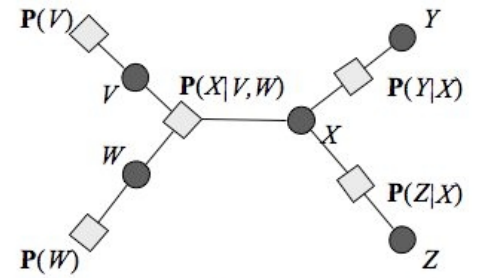
Factor graphs generalize this to decomposition of arbitrary functions of N variables – e.g., $\mathbf{F}(V,W,X,Y,Z) = \mathbf{F}_1(V,W,X)\mathbf{F}_2(X,Y,Z)\mathbf{F}_3(Z)$ – for efficient marginal computation via sums and products. The function is represented as a bipartite graph, with a *variable node* for each variable, a *factor node* for each function use, and undirected links between factors and their variables (Figure 2).



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

Figure 1: Example Bayesian Network

The core inference algorithm for factor graphs is the *sum-product algorithm*, also called the *summary-product* or *belief-propagation algorithm*. It operates by message passing along links. A message from a source node to a target node along a link always 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 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, and then all variables other than the target variable are 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 wherever possible.



(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

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 as you move up the tree. Beyond this simplest case, the algorithm works iteratively by sending output messages from nodes as input messages are received. 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, correct answers are not guaranteed nor is termination (unless arbitrary termination conditions are added). However, the algorithm does often work quite well in practice, as has been demonstrated most notably 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 computation of Most Probable Explanations (MPEs). OR-AND is also fine, as are various other operation pairs.

To improve the efficiency of the algorithm, there are various optimizations that can be applied, and alternative algorithms that can be used (such as survey propagation [Mézard *et al*., 2002] and Monte Carlo sampling [Bonawitz, 2008]). Recently, a connection has been drawn between factor graphs and statistical mechanics, revealing that the sum-product algorithm minimizes the *Bethe free energy*, and yielding additional innovations [Yedidia *et al*., 2005].

3 Factor Graphs for Cognitive Architecture

The key question for us is whether factor graphs can yield a uniform implementation level for understanding and exploring cognitive architecture – the hypothetical narrow waist in the hourglass – while ultimately yielding novel architectures that are more uniform, unified, and functional. The existing work on hybrid mixed methods is encouraging. So too is recent work on languages for mixed probabilistic and logical reasoning. Blaise [Bonawitz, 2008] combines generalized factor graphs with limited relational concepts. FACTORIE [McCallum *et al*., 2008] combines factor graphs with an imperative programming language to support relations and other capabilities. BLOG [Milch *et al*., 2007] and Alchemy [Domingos *et al*., 2006] combine probability and logic via other variants of graphical models.

The particular approach pursued here is to: (1) re-implement existing architectures to help better understand factor graphs, existing architectures, and the implications of implementing architectures via factor graphs; (2) look to go beyond existing architectures by hybridization and simplification, both across architectures and across mechanisms within 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 and its dual status as both a uniform (early on) and a diverse (latest) architecture. This enables starting with an initial uniform core and building towards a more uniform integration of the later diversity. The innermost 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 instantiations of all productions, which then all fire in parallel to modify WM.

The core of this processing is *match*, based on Rete [Forgy, 1982]. Rete uses a discrimination network to sort working memory elements (wmes) to appropriate production conditions and a join network to determine which combinations of matching wmes yield legal production instantiations (while also attending to across-condition variable binding equality). Rete is an efficient algorithm that supports incremental match across cycles and shared match across productions. Most individual productions match efficiently; however, in the worst case, match cost is exponential in the number of conditions [Tambe and Newell 1988].

A design for implementing Rete has been developed, in which factor nodes perform the tests in the discrimination and join networks, variable nodes represent bindings of wmes to production conditions and their combinations – analogous to Rete's α and β *memories* – and message passing is unidirectional (an expression tree) to enable straightforward incremental and shared match (both of which become more problematic with bidirectional message passing).

Rather than imposing Rete a priori on factor graphs, the primary focus to date has instead been on algorithms that arise more naturally from decomposition of production match. Consider Figure 3, which shows a syntactically sugared version of a simple implemented rule. This is not strictly Soar's representation, although it does keep the object-attribute-value scheme, with conditions testing wmes via constants and variables (in angle brackets). The most natural mapping of this production's match 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 overall function is to be decomposed (Figure 4): $\mathbf{P_1}(v_0,v_1,v_2) = \mathbf{C_0}(v_0,v_1)\mathbf{C_2}(v_1,v_2)$.

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

Figure 3: Example Rule

This mapping has been implemented, with the added points that: (a) WM is a 3D, Boolean array – with a dimension each for objects, attributes, and values – in which there is a 1 for every wme in WM and a 0 elsewhere; and (b) a message is a Boolean vector with a 1 for every acceptable binding of the link's variable and a 0 elsewhere. In essence, productions define the graph structure while WM defines distributions over the variables used in factors.

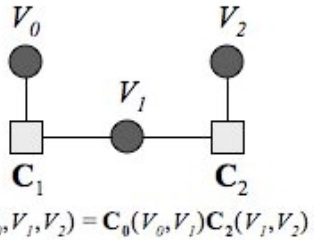


Figure 4: Example Rule Graph

While this demonstrates the initial feasibility of implementing match via factor graphs, it also raises three issues: (1) both working memory and constant tests are hidden within the condition factors; (2) there is no guarantee that all conditions actually match (not in this rule, but elsewhere); and (3) it leads to *binding confusion* [Tambe and Rosenbloom, 1994]. Solutions for all three issues have been implemented, but as the first one doesn't affect correctness – only how much factor graphs are leveraged in the implementation – and the second can't occur in Soar given its production syntax, only the third issue is discussed here.

Binding confusion arises because, instead of maintaining explicit combinations of condition instantiations – as in Rete – the graph independently tracks the legal bindings of each variable (called *instantiationless match* in [Tambe and Rosenbloom, 1994]). 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 and C, v_1 to B and D, and v_2 to Red and Blue, but it can't, for example, distinguish which color (v_2) to associate with object A (v_0), despite the fact that a correct match would clearly mandate Red rather than Blue.

Multiple approaches to this problem are conceivable, divided into whether they alter the match to produce correct combinations or redefine what the match is to compute. To alter the match, we can consider post-extracting the correct combinations [Dechter and Pearl, 1987] or directly implementing Rete. Alternatively, we could redefine match to achieve what is actually produced, and then write rules that yield the desired overall behavior given the new semantics. Or, we could refine this through use of a mixed representation, assigning 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 dispensing with full instantiations. In the process, it eliminates the possibility of 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 (where condition-variable bindings differ while action-variable bindings do not).

This approach modifies variable nodes to represent combinations of production variables rather than solely individual ones. To begin, an ordering is imposed on the production's conditions and actions, and their factor nodes are sequenced accordingly, with a variable node placed between each successive pair of factor nodes. Then, for each production variable, the first and last condition/action using it is determined, and the production variable is added to each variable node between these two factor nodes (Figure 5).

This approach is based on *stretching* in factor graphs, which itself maps onto junction trees [Kschischang *et al.*, 2001]. It has been implemented, and eliminates binding confusion by tracking variable binding combinations as needed.

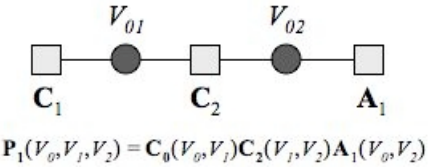


Figure 5: Modified Rule Graph

Since variable nodes may now represent multiple production variables, multi-dimensional arrays result that may be

expensive to process without appropriate optimizations. The most critical optimization is factor rearrangement (Section 2). Without it, the full factor graph for the rule in Figure 3 – incorporating solutions to all three mentioned issues plus the goal memory to be discussed in Section 4, and comprising 8 factor nodes and 8 variable nodes – exhausts heap space before match completes (in LispWorks PE). With factor rearrangement, match takes 1.7 sec.

A second critical optimization leverages the uniformity of the arrays – almost all 0s or 1s – via an N-dimensional generalization of *region quad/octrees* (à la CPT-trees in Bayesian networks [Boutilier *et al*., 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, and match time is reduced by a factor of ~7 (from 1.7 to .25 sec.). This optimization also enables comparing match cost without rearrangement, yielding a factor of ~500 (132 vs. .25 sec.). It will also eventually make sense to compare costs with a conventional Rete matcher, but that is premature with this rough prototype.

One interesting implication of representing WM via trees is the emergence of 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.

4 Conclusion and Next Steps

Despite the increasing trend towards diversity within cognitive architectures, there is still hope for uniformity at the implementation level that could facilitate exploring, understanding and improving existing architectures; and in developing novel ones with increased elegance and broader functionality. Factor graphs are particularly intriguing for this level because they yield a wide diversity of capabilities in a uniform manner. Initial progress has been made towards a reimplementation of the Soar architecture via factor graphs, involving an extension of the understanding of their use in symbol processing to the problem of production match.

The next steps from here are being guided by an overall architectural conceptualization in which perception yields an external distribution for the current world, long-term memory defines a distribution for all possible worlds [Domingos *et al.*, 2006], and WM is the distribution for the current world derived from these by the cognitive inner loop.

The first step is implementing the rest of Soar's inner loop – the *deliberate layer* – where elaboration cycles repeat until quiescence (the *elaboration phase*) followed by a decision. One approach to the elaboration phase is simply to alter WM between cycles, as in conventional production systems. This has been implemented, but a more tantalizing alternative is to arrange the elaboration phase's temporal structure in space rather than time, through the use of a trellis structure. With a trellis, dynamic perceptual and motor processing may be integrated directly into the cognitive inner loop rather than being walled off into a separate I/O system and later cycles within a single phase may retroactively affect earlier ones. Given either approach, *influence diagrams* are a natural first strategy to explore for decisions.

A second step is exploring more uniform approaches to adding semantic and episodic memory based on their appropriate roles in the overall conceptualization. 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 probabilistic view of retrieving the most probable semantic memory element given the probe [Anderson, 1990]. A goal memory – analogous to WM – has been implemented, enabling backwards access to production actions; however, appropriate control of backwards vs. forward processing in the inner loop is still needed, as is limiting retrieval to the most probable element (based on max-product). For episodic memory, two approaches are being evaluated: (1) adding trellis structures in the factor graph; and (2) extending WM to a fourth, temporal dimension.

A third step is making progress on a mixed architecture, combining insights from factor graphs and existing cognitive architectures with results on mixing probabilities and logic (or constraints); and also evaluating the extent to which recently developed mixed languages – such as Alchemy, Blaise, BLOG, and FACTORIE – can be leveraged.

Still, this is only the beginning. It also will be critical to implement and integrate other cognitive capabilities, such as planning, learning, emotion, social cognition, language, perception and motor control – and to reexamine the implementation of a broader range of architectures – all towards developing both a uniform implementation level – the cognitive hourglass's narrow waist – and better architectures.

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. I also owe much to [Kschischang *et al.*, 2001], which first attracted my attention to factor graphs, was critical in understanding them, and is the basis for much of the general material here on them.

References

[Anderson, 1990] John R. Anderson. *The Adaptive Character of Thought*. Erlbaum, Hillsdale, NJ, 1990.

[Anderson, 1993] John R. Anderson. *Rules of the Mind*, Erlbaum, Hillsdale, NJ, 1993.

[Bonawitz, 2008] Keith A. Bonawitz. *Composable Probabilistic Inference with Blaise*. PhD thesis (MIT-CSAIL-TR-2008-044), MIT, July 2008.

[Boutilier *et al*., 1996] Craig Boutilier, Nir Friedman, Moises Goldszmidt and Daphne Koller. Context-specific independence in Bayesian networks. In *Proceedings of the Twelfth Conference on Uncertainty in Artificial Intelligence*, pages 115-123, Portland, OR, August 1996. Morgan Kaufman.

[Dechter and Mateescu, 2003] Rina Dechter and Robert Mateescu. A simple insight into iterative belief propagation's success. In P*roceedings of The Nineteenth Conference on Uncertainty in Artificial Intelligenc*e, pages 175-183, Acapulco, Mexico, July 2003. Morgan Kaufman.

[Dechter and Pearl, 1987] Rina Dechter and Judea Pearl. Network-based heuristics for constraint-satisfaction problems. *Artificial Intelligence*, 34(1): 1-38, 1987.

[Deering, 1998] Steve Deering, *Watching the waist of the protocol hourglass*, Keynote address at ICNP '98, October 1998.

[Domingos, In press] Pedro 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.

[Domingos *et al.*, 2006] Pedro Domingos, Stanley Kok, Hoifung Poon, Matt Richardson, and Parag Singla. Unifying logical and statistical AI. In *Proceedings of the Twenty-First National Conference on Artificial Intelligence*, pages 2-7, Boston, MA, July 2006. AAAI Press.

[Drost and Singer, 2003] Robert J. Drost and Andrew W. Singer. Image segmentation using factor graphs. In *Proceedings of the 2003 IEEE Workshop on Statistical Signal Processing*, pages 150-153, October 2003.

[Forgy, 1982] Charles L. Forgy. "Rete: A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem". *Artificial Intelligence*, 19(1): 17-37, 1982.

[Gogate and Dechter, 2005] Vibhav Gogate and Rina Dechter. Approximate Inference Algorithms for Hybrid Bayesian Networks with Discrete Constraints. In *Proceedings of the 21st Conference on Uncertainty in Artificial Intelligence*, pages 209-216, Edinburgh, Scotland, July 2005.

[Jilk *et al*., 2008] SAL: An explicitly pluralistic cognitive architecture. *Journal of Experimental and Theoretical Artificial Intelligence*, 20(3): 197-218, September 2008.

[Kschischang *et al*., 2001] Frank R. Kschischang, Brendan J. Frey, Hans-Andrea Loeliger. Factor graphs and the sum-product algorithm. *IEEE Transactions on Information Theory*, 47(2): 498-519, February 2001.

[Laird, 2008] John E. Laird. Extending the Soar cognitive architecture. In *Artificial General Intelligence 2008: Proceedings of the First AGI Conference*, Memphis, TN, March 2008. IOS Press.

[Laird and Rosenbloom, 1996] John E. Laird and Paul S. Rosenbloom. The evolution of the Soar cognitive architecture. In D. M. Steier. and T. M. Mitchell (Eds.), *Mind Matters: A Tribute to Allen Newell*, pages 1-50. Erlbaum, Mahwah, NJ, 1996.

[Laird *et al*., 1986] John E. Laird, Paul S. Rosenbloom, Allen Newell. Chunking in Soar: The anatomy of a general learning mechanism. *Machine Learning,* 1(1): 11-46, March 1986.

[McCallum *et al*., 2008] Andrew McCallum, Khashayar Rohanemanesh, Michael Wick, Karl Schultz, Sameer Singh. FACTORIE: Efficient probabilistic programming via imperative declarations of structure, inference and learning. In *Proceedings of the NIPS workshop on Probabilistic Programming*, Vancouver, Canada, 2008.

[Mézard *et al*., 2002] Marc Mézard, Giorgio Parisi, and Riccardo Zecchina. Analytic and algorithmic solution of random satisfiability problems. *Science*, 297: 812-815, 2002.

[Milch *et al*., 2007] Brian Milch, Bhaskara Marthi, Stuart Russell, David Sontag, Daniel L. Ong, and Andrey Kolobov. BLOG: Probabilistic models with unknown objects. In L. Getoor and B. Taskar, (Eds.) *Introduction to Statistical Relational Learning*, pages 373-398. MIT Press, Cambridge, MA, 2007.

[Minsky, 1985] Marvin L. Minsky. *The Society of Mind*. Simon & Schuster, New York, NY, 1985.

[Newell, 1990] Allen Newell. *Unified Theories of Cognition*. Harvard University Press, Cambridge, MA, 1990.

[O'Reilly and Munakata, 2000] Randall C. O'Reilly and Yuko Munakata. *Computational Explorations in Cognitive Neuroscience: Understanding the Mind by Simulating the Brain*. MIT Press, Cambridge, MA, 2000.

[Pearl, 1988] Judea Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufman, San Mateo, CA, 1988.

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

[Rosenbloom *et al*., 1993] Paul S. Rosenbloom, John E. Laird, and Allen Newell. *The Soar Papers: Research on Integrated Intelligence*. MIT Press, Cambridge, MA, 1993.

[Rosenbloom *et al*., 1991] Paul S. Rosenbloom, Allen Newell, and John E. Laird. Towards the knowledge level in Soar: The role of the architecture in the use of knowledge. In Kurt VanLehn (Ed.), *Architectures for Intelligence*, pages 75-111. Erlbaum, Hillsdale, NJ, 1991.

[Tambe and Newell, 1988] Milind Tambe and Allen Newell. Some chunks are expensive. In Proceedings of the Fifth International Conference on Machine Learning, pages 451-458, 1988.

[Tambe and Rosenbloom, 1994] Milind Tambe and Paul S. Rosenbloom. Investigating production system representations for non-combinatorial match. *Artificial Intelligence*, (68)1: 155-199, 1994.

[Yedidia *et al*., 2005] Jonathan S. Yedidia, William T. Freeman, and Yair Weiss. Constructing free-energy approximations and generalized belief propagation algorithms. *IEEE Transactions on Information Theory*, 51(7): 2282-2312, 2005.