To appear in *Proceedings of the First International Conference on Biologically Inspired Cognitive Architectures* (BICA), 2010.

Implementing First-Order Variables in a Graphical Cognitive Architecture

Paul ROSENBLOOM
Department of Computer Science and Institute for Creative Technologies
University of Southern California
12015 Waterfront Dr., Playa Vista, CA 90094
Rosenbloom@usc.edu

Abstract. Graphical cognitive architectures implement their functionality through localized message passing among computationally limited nodes. First-order variables – particularly universally quantified ones – while critical for some potential architectural mechanisms, can be quite difficult to implement in such architectures. A new implementation strategy based on message decomposition in graphical models is presented that yields tractability while preserving key symmetries in the graphs concerning how quantified variables are represented and how symbols, probabilities and signals are processed.

Keywords. Cognitive architecture, graphical models, first-order variables

Introduction

In logic, variables lift sentences from propositional to first order, influencing their truth-values when combined with existential and universal quantifiers. In cognitive architectures, variables enhance the generality of long-term memory by leaving aspects unspecified at storage time. Variables are bound at access time, via match to working memory, with their reuse enabling value propagation while ensuring binding consistency. Quantification occurs, but rather than affecting truth-values, it modulates binding generation; existential yields one binding while universal yields all bindings.

Although implementation of first-order variables is easy in traditional symbolic architectures, it is quite challenging in *graphical architectures* that are characterized by restricting architectural representation and reasoning to networks of simple locally connected units. Conventional neural networks, for example, limit inter-unit messages to numbers, while units compute outputs that are simple nonlinear transformations of their inputs. This article presents a new approach to implementing first-order variables in graphical cognitive architectures that takes inspiration from the Rete match algorithm [1] to combine tractability with preservation of key architectural symmetries.

1. Graphical Cognitive Architectures

Graphical cognitive architectures are of great interest for at least two reasons: (1) they provide a path for grounding cognitive behavior in the network structure of the brain [2, 3]; and (2) they provide the hope of uniformly implementing and integrating state-of-the-art symbol, probability and signal processing to enable, for example, abstract reasoning in uncertain situations and tight coupling of cognition with perception and motor control [4]. The first reason is well understood in the cognitive architecture community, and is part of the core motivation for work on biologically inspired cognitive architectures. The second reason has been less well understood on the whole, although fragments of it have been accreting over time. Signal processing has, for example, long been a forte of neural networks [5], while some models can also be related to probability and symbol processing [6, 7].

In general, neural networks are aimed directly at the first motivation, while progressively being extended towards the second. In contrast, the distinct-but-overlapping class of *graphical models* is aimed directly at the second motivation, while extending towards the first. Rather than focusing on neurological analogies, graphical models emphasize efficient computation with complex multivariate functions, by factoring them into products of simpler subfunctions – based on forms of independence among the variables – and then mapping these products into graphs that compute marginals of variables (i.e., distributions over the values of variables) and the most probable explanation, or MPE (i.e., the optimal combination of values over all of the variables). Bayesian networks are graphical models that provide state-of-the-art techniques for probabilistic reasoning [8]. Markov random fields and hidden Markov models are graphical models that provide state-of-the-art techniques for perception [9]. Graphical models are also making increasing inroads into symbol processing, where the challenge of lifting them to incorporate first-order variables is an active research topic, e.g. [10]. What is most compelling about graphical models from an architectural perspective is the potential for identifying a single generic flavor, with a single processing algorithm, that will uniformly yield effective and efficient behavior across signal, probability and symbol processing. The hope for neural grounding ultimately stems from the fact that a number of neural network models – such as radial basis functions and supervised Boltzmann machines – already map directly onto them [6].

Drawn by this potential, I have been rethinking cognitive architectures from the ground up based on graphical models. This has involved, for example, reconsidering the kinds of mechanisms in Soar 9 [11] and ACT-R [2], and whether it is possible to broaden their overall functionality and coverage while increasing their simplicity and uniformity. What is currently implemented falls short of a complete architecture, but it does provide a *hybrid* (discrete and continuous) *mixed* (Boolean and Bayesian) *memory architecture* embodying a rule-based procedural memory, semantic and episodic declarative memories, and a constraint memory [12]. Work is also in progress on such additional capabilities as decision making, spatial imagery, and learning.

This implementation is based on *factor graphs*, a flavor of graphical model that stems from coding theory, plus the *summary product algorithm* [13]. Factor graphs can decompose general multivariate functions, rather than just the probability distributions dealt with by Bayesian networks. They are undirected bipartite graphs composed of nodes for both variables and factors. A variable links to a factor if the variable is used in the factor's subfunction. The summary product algorithm passes messages among nodes to compute marginals and MPEs. A message between a variable and a factor specifies a distribution over the possible values of the variable. For now, just think of a probability distribution over the domain of the variable. Messages arriving at variables are combined into output messages via a pointwise product, which is like an inner product – where values of corresponding elements are multiplied – but with the result remaining a vector/distribution rather than being reduced to a single value. Each factor also defines a subfunction that is included in the pointwise product of its input messages before all of the variables not in the output message are summarized out, either by summation (for marginals) or maximization (for MPE).

In implementing the memory architecture, the question of how to deal with first-order variables turned out to be central. Existential variables that return the single best value are critical for cue-based retrieval of the best answers from the semantic and episodic memories. Universal variables are essential to the workings of the rule memory, which follows Soar's strategy of determining all legal combinations of variable bindings rather than ACT-R's strategy of limiting rule match to one combination. Universal variables are also needed for the constraint memory to return all consistent bindings, and can play useful roles in declarative memory; for example, in retrieving attribute and category information from semantic memory in parallel for multiple objects in working memory. Although the focus in the remainder of this article is on implementing first-order variables in cognitive architectures based on graphical models, given the overlap between graphical models and neural networks it is hoped that these results will ultimately prove useful more broadly as well.

2. Implementing First-Order Variables

Let's begin with the simple case of existential variables that return the single best value, and then shift to the much harder one of universal variables. In factor graphs, existential variables map directly onto variable nodes. Each message to such a node specifies a distribution over the possible values of the variable. If maximization is used to summarize out variables that are unneeded in the outputs of factor nodes, then the MPE – i.e., the single combination of bindings to the variables with the highest joint value – is computed. This is used in episodic memory to retrieve the stored episode that best combines recency – based on an exponentially decaying temporal variable – with match to the cues. If summation is used instead, marginal distributions over the individual variables are computed. This is used in semantic memory to predict the categorization given the cues and to retrieve the best values for all uncued attributes.

Universal variables are the core hard aspect of implementing first-order variables in graphical architectures. Consider all-binding rule match, as in Soar. Instead of messages delineating whether individual domain elements yield the best value, messages must now convey information about which sets of domain elements may be legal. Thus, the domain of the corresponding variable consists of subsets of the quantified variable's domain elements rather than individual such elements. For example, if the domain of a quantified variable is <A, B>, then the domain of the variable node is <{}, {A}, {B}, {A B}>. A message to such a node would be a distribution over these four elements, such as <.4, .2, .1, .3>. Although such an approach is correct, it requires quite a different representation for universal variables from that used for existential variables. Even worse, it is computationally intractable; if the quantified variable's domain is of size N, the variable node's domain is of size 2^N.

The obvious response to this intractability is to seek a decomposition strategy that avoids the combinatorics. In Markov logic, a mix of first-order logic and probabilities [10], the combinatoric variable node is decomposed into N Boolean variable nodes, one for each of the N domain elements of the quantified variable. By propositionalizing both the variable and the graph, each variable node and message now reflects the probability that one original domain element is bound to the quantified variable. To continue with our simple example, there are now two variables, A and B, each with two domain elements: <T, F>. Messages are distributions over these variable domains, such as A:<.2, .8> and B:<.4, .6>. Such an approach reduces the number of elements of concern from 2^N to 2N in general. However, interactions among values of multiple variables must also be instantiated as further Boolean nodes; so, if there were a second quantified variable with the domain <C, D> that interacts with the first, nodes would need to be added for A&C, B&C, A&D and B&D. This effectively performs key aspects of symbol processing – such as the determination of partial instantiations in rule match – during graph compilation, while propagating the effects of propositionalization to combinations of values. In the process it reintroduces the issue of lifting, although now for efficiency rather than expressibility. The approach also breaks two key forms of symmetry: (1) between existential and universal quantifiers, with the former remaining first order while the latter are propositionalized; and (2) between the processing of symbols and the processing of signals and probabilities, with (much of) the former occurring during graph construction and the latter occurring during message passing in the constructed graph.

The alternative decomposition strategy presented here preserves these symmetries, but at the cost of introducing other forms of complexity. It was inspired by the Rete match algorithm for rules. Rete is a message passing algorithm that uses a discrimination network to determine which working memory elements match individual rule conditions, plus a join network to determine which combinations of these elements match particular combinations of conditions. Rete's messages represent partial instantiations of rules, but they can be stripped down to just the variable bindings they embody without loss of information. Rather than specifying a vector of probabilities over a variable's domain elements, such a message specifies a Boolean vector. Each value in the vector acts a bit like its own Boolean variable over an individual domain element, and each message says something about all possible bindings of the quantified variable. For our simple example, the domain of the variable node is now just <A, B>, with a typical message being <1, 0>, saying that A is a possible value but B is not. In general, this approach thus uses only a single variable node with a domain of size N to implement a universal variable. However, when multiple variables interact, variable nodes in the join network must now represent their combination, yielding domains that are cross products of the individual variable domains. For the two-variable example above, there is now a single 2D variable node with a 2x2 domain covering combinations of the two variables' domain elements.

This approach enables retaining first-order universal variables in the graph because decomposing the domains and messages enables the nodes themselves to remain unitary. To avoid total message propositionalization – where all (combinations of) variable values are listed explicitly – and to maintain the symmetry with probability and signal processing, all messages are actually represented as N dimensional continuous functions that are approximated in a piecewise-linear fashion [12]. For the current Boolean case, this means that regions of the function that are a constant (either 0 or 1) remain undifferentiated. Arbitrary continuous functions are more complex in general than the single-valued messages typical in neural networks, but they are consistent with the overall expressibility of messages in graphical models and may perhaps ultimately be related to something appropriate over populations of neurons.

This approach to decomposition is provably correct for the special case of functions with a Boolean (0/1) range, as are used here. The proof depends on mapping between this factored representation and the original combinatoric representation; in particular, a message with 1s for a set of elements in the factored representation maps to a combinatoric message in which all subsets of these elements are 1. To yield correct results, product and summarization must compute equivalent values when applied to both the factored representation and its corresponding combinatoric representation. Pointwise product of factored vectors yields 1s only where all inputs are 1s, computing the intersection of the legal elements in the inputs. Pointwise product of combinatoric vectors performs essentially the same computation by generating 1s for those subsets of values legal in all inputs. So, mapping factored vectors to combinatoric vectors, computing their pointwise product, and then mapping them back yields the same result as is computed directly on factored vectors. The same is also true for maximization; the result is a 1 in both cases if there is at least one legal binding in a vector. Summation, however, does yield different results, with factored vectors yielding the number of possible bindings and combinatoric vectors yielding the number of subsets of bindings. Fortunately summation isn't needed with Boolean vectors, since all that really matters is whether there is at least one possible binding.

Correct results are not, however, guaranteed when probabilities over individual elements are used rather than Boolean values. Product still yields the correct values, but neither sum nor max does. Thus, with this factoring strategy, probabilities can be used for existential variables but not for universal ones. The latter must be Boolean.

In addition to ensuring the correctness of the summary product algorithm, an equally important question is whether the necessary subfunctions can be implemented as factors in the decomposed representation. In rule match, the subfunctions of interest are filters. A constant test eliminates elements that might otherwise match a condition. A join node eliminates partial instantiations lacking compatible variable bindings. Such filters are many-one functions that can be expressed directly in the combinatoric representation, with a value of 1 for all pairings of the filter's input and output values that meet the criteria. For a forward message through a filter, the output receives all subsets of values of the input variable compatible with the filter. In the reverse direction, indeterminacy results for all values that might have been filtered out when going forward. In the decomposed representation, the forward direction is implemented via a factor function containing a 1 for anything that passes the filter and a 0 for everything else. However, a backwards message through such a function does the same filtering rather than generating the expected ambiguity. In rule systems, messages only proceed in one direction – from working memory through conditions to actions – and thus this inverse function is not relevant, but were it needed, the summary product algorithm could be extended to use a different function for each output of a factor in service of preserving the intended decompositional semantics of the graph.

Beyond filters, there are subfunctions that cannot be expressed as decomposed factor functions for computation via summary product. For these cases, special-purpose one-way code is used that, when applied to the factored representation, yields the result that would be obtained from the combinatoric representation. Computing the inverse of a variable, as occurs in implementing negated conditions, is a simple example. Given a Boolean message about the elements of a variable, it should generate an inverse message with 1s replaced by 0s and 0s by 1s. The true factor function should have a 1 wherever the input and output messages are inverses and a 0 otherwise. This is easy to specify in the combinatoric representation, but not in the decomposed representation. Instead, inversion is implemented by special-purpose code that bypasses summary product within the node to directly generate the inverse output from the input. This code works the same in both directions here, but in other cases – such as when computing disjunction over messages – different code is required for each output. The overall network still computes the same value as would summary product on the combinatoric representation, but special purpose code is used for tractability.

3. Conclusion

Graphical cognitive architectures provide hope of grounding their mechanisms in the workings of the brain while yielding a uniform and tightly integrated combination of state-of-the-art capabilities across symbols, probabilities and signals. This article has presented a new approach to handling a key yet challenging capability in such architectures: first-order variables. The result involves some complexity, and requires special-purpose optimizations for some aspects, but in turn it enables coping with such variables – even when universally quantified – in first-order graphs, while keeping the processing of symbols aligned with that of signals and probabilities. This approach has been successfully applied to implement a hybrid mixed memory architecture [12].

Acknowledgement

This effort has been sponsored by the USC Institute for Creative Technologies and the U.S. Army Research, Development, and Engineering Command (RDECOM). Statements and opinions expressed do not necessarily reflect the position or the policy of the United States Government, and no official endorsement should be inferred.

References

[1] Forgy, C. L. (1982). Rete: A fast algorithm for the many pattern/many object pattern match problem. *Artificial Intelligence*, *19*, 17-37.

[2] Anderson, J. R. (2009). *How Can the Human Mind Occur in the Physical Universe?* Oxford: Oxford University Press.

[3] Smolensky, P. & Legendre, G. (2006). *The Harmonic Mind: From Neural Computation to Optimality-Theoretic Grammar*. Cambridge, MA: The MIT Press.

[4] Rosenbloom, P. S. (2010). Rethinking cognitive architecture via graphical models. *Cognitive Systems Research*. In press.

[5] Hu, Y. H. & Hwa, J.-N. (2002). *Handbook of Neural Network Signal Processing*. Boca Raton, FL: CRC Press.

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

[7] Lallement, Y., Hilario, M. & Alexandre, F. (1995). Neurosymbolic integration: Cognitive grounds and computational strategies. In M. DeGlas and Z. Pawlak (Eds.), *World Conference on the Fundamentals of Artificial Intelligence*, 193-203.

[8] Koller, D. & Friedman, N. (2009). *Probabilistic Graphical Models*. Cambridge, MA: MIT Press.

[9] Li, S. Z. (2009). *Markov Random Field Modeling in Image Analysis*. London: Springer.

[10] Domingos, P. & Lowd, D. (2009). Markov Logic: An Interface Layer for Artificial Intelligence. Morgan & Claypool.

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

[12] Rosenbloom, P. S. (2010). Combining procedural and declarative knowledge in a graphical architecture. In *Proceedings of the 10th International Conference on Cognitive Modeling*.

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