Computer Science Department Technical Report
University of California
Los Angeles, CA 90024-1596

STORING AND GENERALIZING MULTIPLE INSTANCES WHILE MAINTAINING KNOWLEDGE-LEVEL PARALLELISM

R. A. Sumida
M. G. Dyer

August 1989
CSD-899999

Storing and Generalizing Multiple Instances while Maintaining Knowledge-Level Parallelism*†

Ronald A. Sumida
Michael G. Dyer
Artificial Intelligence Laboratory
Computer Science Department
University of California
Los Angeles, CA, 90024

Abstract

One of the primary problems in knowledge representation and learning is determining how multiple instances of concepts should be organized and represented. Symbolic approaches, such as semantic networks, have been successful at representing structured knowledge for parallel access. However, such approaches have had difficulty organizing multiple instances for automatic generalization and efficient retrieval. Parallel distributed processing systems (PDP) appear to offer a solution to these problems. Unfortunately, current PDP models have not yet been able to satisfactorily represent complex knowledge structures and they remain sequential at the knowledge level. This paper presents an approach which stores multiple instances in ensembles of PDP units and organizes the ensembles in a semantic network for parallelism and structure. Thus, the best features of both styles of representation are obtained.

1 Introduction

One of the central problems in knowledge representation and learning is deciding how to represent and organize multiple instances in memory. Are people able to store and retrieve every instance of an event, such as walking somewhere? Since there are enormous numbers of walking events done by a given person, or done by others and perceived by that person, retrieving every one is nearly impossible. Consequently, people must be organizing and generalizing multiple instances in some manner.

2 Previous Approaches

Previous approaches to knowledge organization and generalization can be divided into two levels, shown in Figure 1.

*This research is supported in part under a contract to the second author by the JTF program of the DoD, monitored by JPL, and by a grant from the ITA Foundation.

†To appear in the Proceedings of the Eleventh International Joint Conference on Artificial Intelligence, Detroit, MI, August, 1989.

What follows is a review of symbolic and PDP approaches to the multiple instances problem.

(1) *Symbolic Systems*: When a new event is encountered, a new token is created to represent it. For a limited number of events, storing every one does not pose a major problem. For large amounts of knowledge, similar events can be grouped together and discriminated by their differences, e.g. [Kolodner, 1984]. Unfortunately, this approach has a number of problems. First, it suffers from combinatorial explosion since there are many ways that shared features can be combined. Second, distinct tokens are needed as a final discriminant of each concept (i.e., the leaves of the tree). Third, an evaluative symbol processing mechanism is needed that decides (a) when to create a symbol, (b) how long to retain it and (c) where to index it. Finally, since each leaf is completely distinct, memory confusions do not naturally arise as a consequence of the representation. In human memory, however, confusions often do occur [Bower *et al.*, 1979].

(2) *Parallel Distributed Processing*: PDP Systems represent each concept over shared weights in a connectionist network [Rumelhart and McClelland, 1986]. A new instance is encoded by changing link weights. Some instances will result in large weight changes and will therefore be remembered. Others, however, will change the weights only slightly and will be very difficult or impossible to recall.

In general, distributed representations address the shortcomings of symbolic systems in handling multiple instances. They avoid the combinatoric problems by having the same unit participate in the representation of multiple concepts. They also naturally account for memory confusions because concepts that share a number of features will have a very similar representation. Thus, when an attempt is made to access a concept from its features, a similar pattern representing another concept may sometimes be recalled.

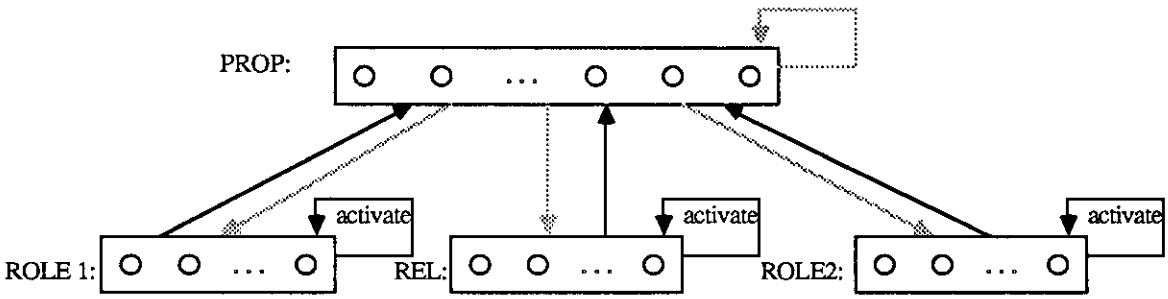
Unfortunately, distributed representations have not yet been successful at encoding structured knowledge while maintaining parallelism at the knowledge level. For example, Hinton [1981] has proposed a method for encoding semantic networks in a single PDP architecture (Figure 2) by representing all knowledge as triples of the form (Role1 Relation Role2).

Hinton's entire network is divided into four ensembles of units: one for each component of a triple and one

Level	System	Multiple Instances	Automatic Generalization	Knowledge-Level Parallelism	Structure w/o Crosstalk
Macro	Symbolic/Localist	-	-	+	+
Micro	Distributed (PDP)	+	+	-	-

"+" indicates support for that feature, "-" indicates lack of support

Figure 1



Dark arrows indicate fully connected random (fixed) weights.
Grey arrows indicate fully connected modifiable weights.
Units in each ensemble self-activate. Circles indicate PDP units.

Figure 2

which represents the entire proposition (PROP). When the patterns representing the components of a triple are placed into the appropriate ensembles shown above, a random pattern is generated on the units in PROP. This pattern, which represents a *reduced description* of the triple, is associated with the constituents that gave rise to it by adjusting the weights between the PROP units and the other three ensembles. By storing propositions in this manner, the network is able to perform simple inheritance and to complete a triple given two of its constituents. Hinton [1988] proposes a generalization of this model, based on the reduced description principle, which can also store embedded structures.

Systems using only a small set of ensembles (e.g. [Hinton, 1988, Touretzky, 1987]) for representing all of their knowledge, suffer from the *Knowledge Parallelism Problem.* That is, these systems are *sequential at the knowledge level* because only one triple can be stored or accessed at a time. Attempts to retrieve multiple triples in parallel will lead to enormous crosstalk problems. Some systems, such as [Dolan and Dyer, 1987, Touretzky and Hinton, 1985], can select a triple in parallel from the space of all available triples. However, they can only do this to the extent that their structures are represented locally (e.g. schema, their constituent roles, and rules, each locally encoded as a subset or node in a winner-take-all network). The loss of knowledge-level parallelism is particularly problematic in constraint-satisfaction inferencing, in which it is essential to pursue multiple paths simultaneously.

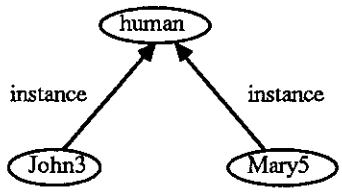
3 A Model for Storing Multiple Instances

We desire a knowledge representation system that: (1) represents structure, (2) is parallel at the knowledge level, (3) handles an enormous number of multiple instances with graceful degradation and automatic generalization, and (4) exhibits memory confusions only in the same situations that people do. Humans usually keep inferred bindings straight while confusing multiple instances of a number of events. For example, if we learn that John shoots Fred, we would like the system to correctly infer and later recall that Fred (not John) bleeds. However, if we present a large number of propositions in rapid succession, (such as a list of different colored shirts different individuals wear) then we would like the system to become confused in the same way that people do.

3.1 General Approach: Parallel Distributed Semantic (PDS) Networks

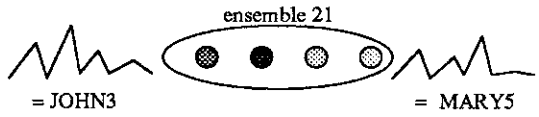
Our approach is to maintain the locality of semantic networks for parallel manipulation and access of structure at the knowledge level, but to make each semantic network

node an ensemble of PDP units to hold multiple patterns of instances. At the macro (localist/semantic network) level, the PDS network holds knowledge of structure with role relations. At the micro (PDP) level, the PDS network holds multiple instances. For example, suppose that we wish to represent two particular humans named John and Mary. In a standard semantic network, the general concept human is represented by a single node, with nodes for John and Mary connected to it by instance (is-a) links (Figure 3a).



Nodes in a standard, Localist Network
Figure 3a

In a PDS network (Figure 3b), humans are represented over a particular ensemble of units (e.g. ensemble 21) and John and Mary are represented as activation patterns over that ensemble.



Ensemble 21 represents a single Semantic Network Node in a PDS Network, currently holding the activity pattern for JOHN3, while also able to represent other instances, such as MARY5.

Figure 3b

The circled dots in the figure indicate the ensemble of units and the jagged lines are suggestive of the patterns of activity that represent John and Mary.

3.2 Issues: Symbol Processing and Knowledge-Level Encoding

Using PDS networks raises two important issues. The first is whether operations needed to handle processing and representation can be properly implemented. Minimally, we need the equivalent of the following operations:

1. Create something like a new symbol. For example, in LISP a Gensym-like function is used to create INGEST.14.
2. Represent and manipulate schemas. Schema operations include: binding roles in one schema to another schema; traversing roles; binding new schemas; etc.
3. Represent rules where variable bindings are propagated between structures. For example, if we have the rule: [X shoots Y] => [Y bleeds], and the bindings X = John and Y = Fred, we would like to propagate the binding Y = Fred from the shoots structure to the bleeds structure.

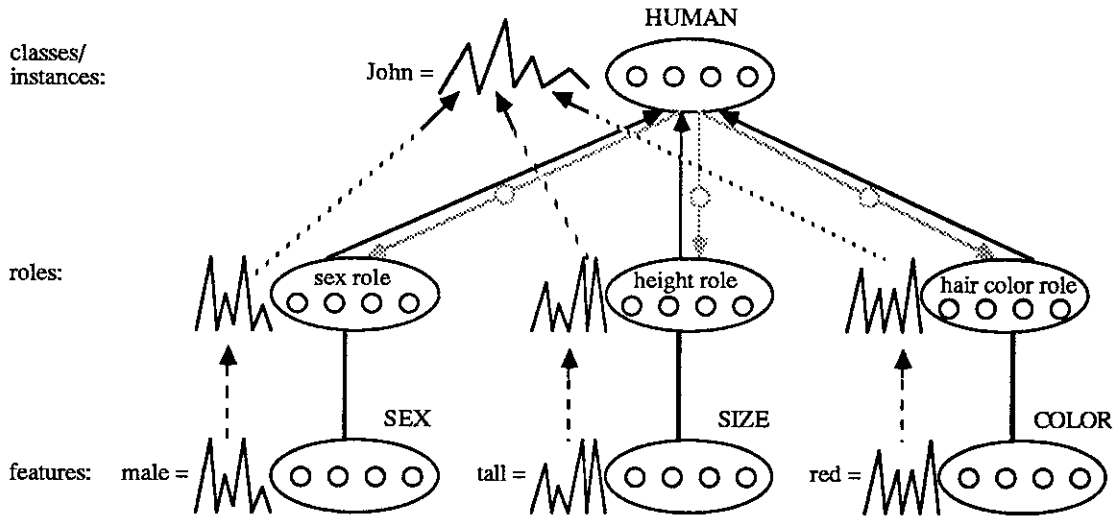
The second issue involves encoding knowledge into the ensembles described earlier. At one extreme, creating a new ensemble for each instance defeats the purpose of using distributed representations and also raises the question (which plagues symbolic systems) of where the new nodes come from. At the other extreme, encoding all schemas and roles into a small set of ensembles causes sequentiality at the knowledge level and massive crosstalk in retrieving concepts. The solution lies somewhere in between.

3.3 The PDS Network Approach

How are new instances added to a PDS network? Suppose our model learns about a new human, John, who is male, tall and has red hair. In order to add John to memory, we will have to generate a new pattern in the human ensemble that will represent John. This is accomplished by placing the patterns for each of John's features into the appropriate ensembles and propagating activation into the human ensemble. The resulting pattern of activity over the human ensemble will then be used to represent John. This process is illustrated in Figure 4. In the figure, the conceptual ensembles (i.e., HUMAN, SEX, SIZE and COLOR) are labelled with capital letters and the role ensembles are indicated by lower-case labels placed inside the oval of units.

The pattern for the first feature, *male*, is placed into the ensemble of units representing *SEX*, loaded into the *sex role* ensemble, and propagated across links with fixed random values into the HUMAN ensemble by the path (dotted line) shown in the figure. The pattern for *tall* is then placed in the ensemble of units representing *SIZE*, propagated across random-valued links from the *height role* ensemble, and combined with the pattern of activity already on the HUMAN nodes. The pattern for *red* is placed into the *COLOR* and *hair color role* ensembles and propagated in a similar fashion. The pattern that is generated from the combined activation of the three features is used to represent John in the HUMAN ensemble.

The process of adding John to memory is not yet complete, because given the pattern for John, we also want to be able to reconstruct his features. We must therefore associate the pattern for John with those for *male, tall* and *red*. Associating these patterns involves changing the variable weights along the path from the HUMAN ensemble to the *sex, height* and *hair color* role ensembles, and can be accomplished using one of a number of generalized learning procedures. For instance, using backpropagation [Rumelhart and McClelland, 1986], the network of Figure 4 can be viewed as a conjunction of three 3-layer networks, with the HUMAN ensemble as the input layer and each of the role ensembles as the output layers (the hidden layers are indicated by the circles on the grey arrows in the figure).



Black arrows = fixed random weighted links.
Grey arrows = modifiable weighted links (single circle indicates hidden layer).
Thick lines = links which propagate a pattern without changing it.
Dotted arrows indicate how activity patterns over ensembles are propagated.

Figure 4

As with Hinton's model, the units within a conceptual ensemble are interconnected by variable weights, which are trained by associating each pattern with itself (this process is referred to as auto-association [Rumelhart and McClelland, 1986]). Thus, when an ensemble is given an unfamiliar pattern that resembles a known one, it will tend to recreate the known pattern. This allows the network to perform pattern completion from partial or noisy input. As an extension to Hinton's model, an intervening hidden layer (not shown in the figure) is included in each conceptual ensemble and is used to interconnect the units within the ensemble. Adding these hidden units allows the ensemble to store much more information, while using fewer units and connections.

3.4 Rules, Schemas, and Bindings in DCAIN

The network structure and operations described above are being used to implement DCAIN, a system designed to store instances of schemas and rules, and to implement static role bindings. For example, suppose DCAIN learns that John told Mary some information. From this, we would like DCAIN to infer both that John knew the information as a precondition to telling Mary, and that Mary now knows the information as a consequence of John's telling her. These inferences are summarized in Figure 5.

The first causal inference (CAUSES.13 in Figure 5) is stored in the architecture shown in Figure 6[1].

Different ensembles of units (ovals in Figure 6) store the communicate (COM), know, causality and enable-

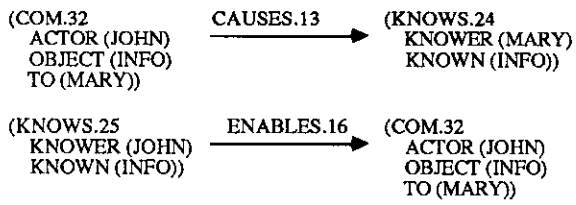
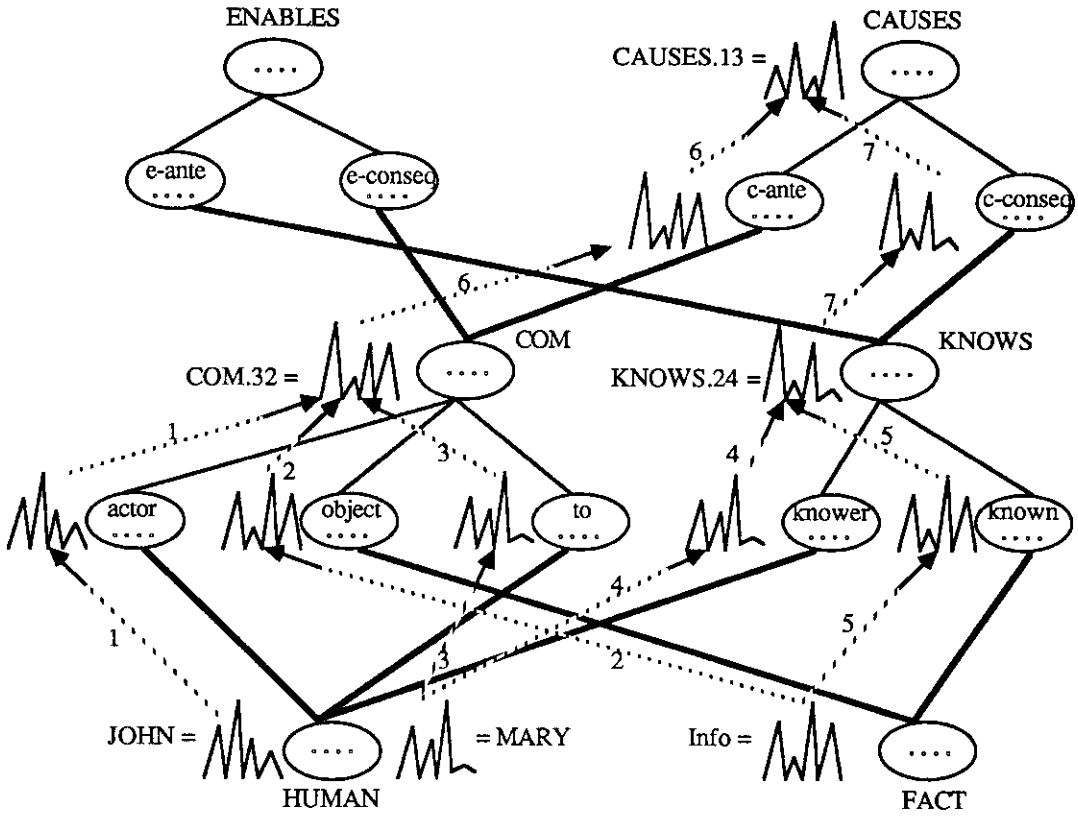


Figure 5

ment schemas, in the following manner: We first represent the proposition (John COM Info TO Mary) by generating a pattern for it in the COM units using the method described in the preceding section. Thus, the pattern for John is propagated from the *actor* ensemble (path 1); the pattern for the information from the *object* ensemble (path 2), and the Mary pattern from the *to* ensemble (path 3). The resulting activity pattern, labelled COM.32, represents the proposition. COM.32 is now (a) associated with its constituents by weight changes between the COM units and the role ensembles, and (b) auto-associated with itself by weight changes within the COM units. Similarly, (Mary KNOWS info) is represented by propagating the pattern for Mary from the *knower* ensemble to KNOWS (path 4) and the pattern for the information from *known* to KNOWS (path 5). The resulting activity pattern, KNOWS.24, is also associated with itself and with the constituents that gave rise to it.

To represent the causality relationship between COM.32 and KNOWS.24, COM.32 is propagated from the antecedent ensemble (*c-ante*) and KNOWS.24 from the consequent ensemble (*c-conseq*) to generate a pat-

[1]For the sake of visual clarity, the black and grey arrows from Figure 4, that connect the conceptual and role ensembles, are replaced by a single, thin line in Figure 6.



Each ensemble is labelled with a name suggesting its role.

Figure 6

tern in CAUSES (paths 6 and 7). The new pattern, labelled CAUSES.13, is then associated with itself and with COM.32 and KNOWS.24. CAUSES.13 now completely represents the first causal relation. In exactly the same manner, (John KNOWS Info) is represented as KNOWS.25, and it and COM.32 are combined in the ENABLES units (Figure 7) to yield a reduced description for ENABLES.16 from Figure 5.

When the network is given (John COM Info TO Mary), it can now make the correct inferences. This proposition is presented by again propagating the patterns for John, the information, and Mary through the COM units, resulting in COM.32. Propagating COM.32, we obtain a pattern over the CAUSES ensemble which is similar to CAUSES.13, and one over the ENABLES ensemble which is much like ENABLES.16. Since partial (or noisy) patterns within an ensemble are trained to complete themselves, the CAUSES.13 and ENABLES.16 patterns will eventually emerge. The emergence of CAUSES.13 means that KNOWS.24 will appear in the *c-conseq* and KNOWS units, which in turn recreates the (Mary KNOWS Info) proposition. Similarly, ENABLES.16 causes KNOWS.25 to emerge, which generates the inference (John KNOWS Info).

As a consequence, if we present [John told Mary Info] to DCAIN, it will reconstruct (1) what was told to Mary, (2) infer John already knew it, and (3) infer what Mary knows. DCAIN accomplishes these reconstructions *without* having to create new nodes to represent each instance, as is required in standard semantic networks such as NETL [Fahlman, 1979].

4 Previous Work, Current Status, and Future Directions

To handle the problems of representing structured knowledge, dynamic role bindings and role propagation, we initially constructed CAIN [Sumida *et al.*, 1988], a marker passing system that uses a localist network and incorporates features of connectionist systems, specifically, link weights, activation values, and thresholds. CAIN propagates both markers and activation to do goal/plan analysis needed for interpretation of natural language input such as "the man hid the pot in the dishwasher when the police came". Here, "pot" must be disambiguated to MARIJUANA by a goal/plan analysis, although "in dishwasher" suggests COOKING-POT.

Currently, we are constructing DCAIN, a distributed version of CAIN based on the principles described here. In DCAIN, patterns evoke other patterns, like [Hinton, 1981, Touretzky, 1987]. Unlike their systems, DCAIN uses many more ensembles to recreate many of the nodes normally existing in semantic networks, but without the instance nodes.

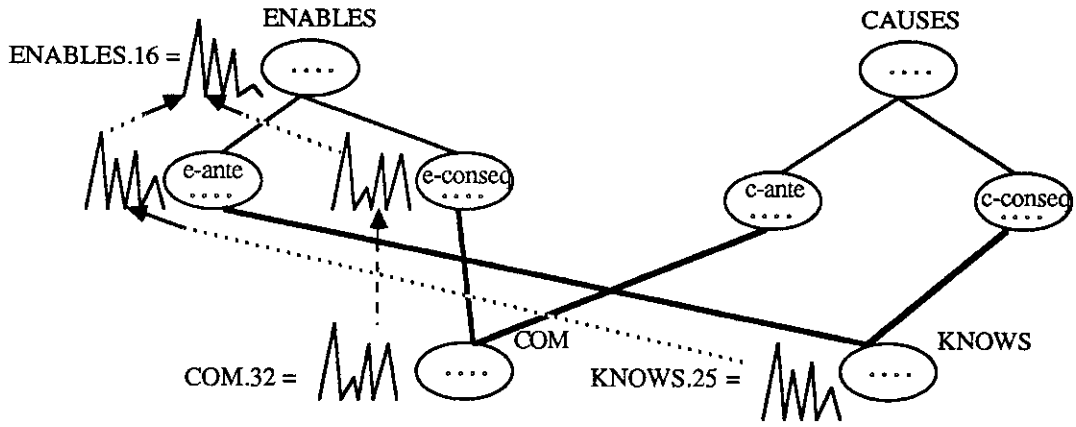


Figure 7

Future directions of research include methods for:

(1) *Propagating dynamic bindings along schema and their roles.* In this paper, we have demonstrated how PDS networks can store *static* role bindings, in which a previously encountered concept is bound to a role of a known proposition. In contrast, *dynamic* bindings involve binding a novel concept and propagating it, unaltered, to the corresponding roles of related structures. We currently assume that the links which propagate dynamic bindings (e.g. the thick black lines in Figures 6 and 7) have exactly the right weight values so that patterns can be passed on without being changed. Dynamic bindings in symbolic architectures are realized by propagating bit patterns, unaltered, along various pathways. Our system needs a method for adjusting the weights so that such patterns are passed along without being modified.

(2) *Forming semantic networks dynamically* through modification of connectivity patterns between units, i.e., both within and across ensembles. One possibility that we are currently investigating involves conscripting new units to form additional ensembles.

5 Conclusions

A parallel distributed semantic (PDS) network called DCAIN, has been designed: (a) to handle the severe combinatorics of storing and retrieving multiple instances while maintaining parallelism at the knowledge level, and (b) to achieve graceful degradation and automatic generalization in the face of a vast number of instances. In DCAIN, ensembles of nodes are macroscopically related like semantic networks and microscopically related like PDP networks. Structured objects such as schemas and rules are represented at the macro-level, while at the micro-level, instances are reconstructed as patterns of activation over PDP ensembles.

References

[Bower *et al.*, 1979] G. H. Bower, J. B. Black, and T. J. Turner. Scripts in memory for text. *Cognitive Psychology*, 11:177–220.

[Dolan and Dyer, 1987] C. P. Dolan and M. G. Dyer. Symbolic Schemata in Connectionist Memories: Role Binding and the Evolution of Structure. In *IEEE First International Annual Conference on Neural Networks*, San Diego, 1987.

[Fahlman, 1979] S. E. Fahlman. *NETL: A System for Representing and Using Real-World Knowledge.* MIT Press, Cambridge, Massachusetts, 1979.

[Hinton, 1981] G. E. Hinton. Implementing Semantic Networks in Parallel Hardware. In *Parallel Models of Associative Memory*, Lawrence Erlbaum, Hillsdale, NJ, 1981.

[Hinton, 1988] G. E. Hinton. Representing Part-Whole Hierarchies in Connectionist Networks. In *Proceedings of the Tenth Annual Conference of the Cognitive Science Society*, Montreal, 1988.

[Kolodner, 1984] J. L. Kolodner. *Retrieval and Organizational Strategies in Conceptual Memory.* Lawrence Erlbaum, Hillsdale, NJ, 1984.

[Rumelhart and McClelland, 1986] D. E. Rumelhart and J. L. McClelland. *Parallel Distributed Processing*, Volume 1. MIT Press, Cambridge, Massachusetts, 1986.

[Sumida *et al.*, 1988] R. A. Sumida, M. G. Dyer, and M. Flowers. Integrating Marker Passing and Connectionism for Handling Conceptual and Structural Ambiguities. In *Proceedings of the Tenth Annual Conference of the Cognitive Science Society*, Montreal, 1988.

[Touretzky, 1987] D. S. Touretzky. A Distributed Connectionist Representation for Concept Structures. In *Proceedings of the Ninth Annual Conference of the Cognitive Science Society*, Seattle, 1987.

[Touretzky and Hinton, 1985] D. S. Touretzky and G. E. Hinton. Symbols Among the Neurons: Details of a Connectionist Inference Architecture. In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, Los Angeles, 1985.