DESCARTES: Development Environment for Simulating Hybrid Connectionist Architectures*

Trent E. Lange, Jack. B. Hodges, Maria. E. Fuenmayor, Leonid. V. Belyaev

Computer Science Department
University of California, Los Angeles

Abstract

The symbolic and subsymbolic paradigms each offer advantages and disadvantages in constructing models for understanding the processes of cognition. A number of research programs at UCLA utilize connectionist modeling strategies, ranging from distributed and localist spreading-activation networks to semantic networks with symbolic marker passing. As a way of combining and optimizing the advantages offered by different paradigms, we have started to explore hybrid networks, i.e. multiple processing mechanisms operating on a single network, or multiple networks operating in parallel under different paradigms. Unfortunately, existing tools do not allow the simulation of these types of hybrid connectionist architectures. To address this problem, we have developed a tool which enables us to create and operate these types of networks in a flexible and general way. We present and describe the architecture and use of Descartes, a simulation environment developed to accomplish this type of integration.

Introduction and Motivation

Within the connectionist approach there are three paradigms, each having its own advantages and disadvantages: Distributed Connectionist Networks (DCNs), Localist Connectionist Networks (LCNs), and Marker-Passing Networks (MPNs).

DCNs (such as the models in [Rumelhart & McClelland, 1986]) use simple, neuron-like processing elements which represent knowledge as distributed patterns of activation. DCNs, sometimes known as *Parallel Distributed Processing* or *subsymbolic* models, are interesting because they have learning rules that allow stochastic category generalization, they perform noise-resistant associative retrieval, and they exhibit robustness to damage. Distributed models, however, have (so far) been sequential at the knowledge level, lacking both the structure needed to handle complex conceptual relationships and the ability to handle dynamic variable bindings and to compute rules.

LCNs (as exemplified by the models of [Waltz & Pollack, 1985] and [Shastri, 1988]) also use simple, neuron-like processing elements with numeric activation and output functions, but represent knowledge using semantic networks of conceptual nodes and their interconnections. Unlike DCNs, localist networks are parallel at the knowledge level and have structural relationships between concepts built into the connectivity of the network. Unfortunately, they lack the powerful learning and generalization capabilities of DCNs. They also have had difficulty with dynamic variable bindings and most other capabilities of symbolic models.

MPNs (as exemplified by the models of [Charniak, 1986] and [Hendler, 1988]) also represent knowledge in semantic networks and retain parallelism at the knowledge level. Instead of spreading numeric activation values, MPNs propagate symbolic markers, and so support the variable binding necessary for rule application, while preserving the full power of symbolic systems. On the other hand, they do not possess the learning capabilities of DCNs or exhibit the inherent evidential constraint-satisfaction capabilities of LCNs.

Hybrid Connectionist Models

Research at UCLA has spanned the range from subsymbolic to symbolic connectionist models [Dyer, 1989]. A number of us have begun to construct hybrid architectures which use what we term Multiple Interacting Networks, or MINs, heterogeneous connectionist networks that communicate via shared elements. A neurophysiological approach [Nenov & Dyer, 1988] effectively uses MINs for visual/verbal association by modeling heterogeneous neuronal characteristics in separate networks. We have also been exploring the use of MINs for higher cognitive tasks, such as planning, creativity, story invention, and political negotiations. In political negotiations research, for instance, MINs are used to simulate the multiple perspectives of negotiating parties.

Another approach is to build models that combine the bottom-up processing features of DCNs with the top-down processing features of LCNs and MPNs. Figure 1 shows **Hiding Pot**, an example wherein elements from each paradigm are combined using MINs. This allows us to approach a problem that would be difficult, if not impossible, using a single paradigm. **Hiding Pot** shows a simplified network built to understand the sentence, *"John put the pot inside the dishwasher because the police were coming."*[1] Network-A in Figure 1 utilizes an MPN to do role-binding and an LCN to activate and combine evidence for individual schemas. These then combine their functionality to support predictions and perform inferencing and disambiguation.

One might also want to combine different connectionist approaches by having separate networks that communicate with each other, where each one performs a different cognitive task. Network-B in Figure 1 is a DCN, trained to recognize words from line segments [McClelland & Rumelhart, 1986, chap. 1]. By integrating these two approaches, we can simulate cognitive processes at the different levels of abstraction necessary for modeling reading and understanding.

Network-A interacts with Network-B through shared lexical nodes. Once a word has been recognized, it passes activation to the concepts related to the word. For example, the node for concept John gets activation from the word node "john" which is shared by both networks. Activation then propagates along the chain of related concepts in the network as contextual evidence for disambiguation. Markers are passed over the role nodes across marker passing links between corresponding roles to represent role-bindings and perform the needed inferencing.

While there are several existing connectionist simulators, none allows the simulation of multiple interacting hybrid networks, as in **Hiding Pot**, that integrate elements from more than one paradigm of connectionist modelling. We have developed the Descartes simulation environment specifically to address this kind of integration. Descartes enables researchers to design, simulate, and debug hybrid connectionist architectures that combine elements of distributed, localist, and marker-passing networks.

Descartes Architecture

Descartes is a package designed for simulating network processing, network interaction, and integration of networks into an overall processing environment. The system consists of two interactive components: network *elements*, such as nodes and links, their associations, and their functionality, and *processing controllers*, which organize network elements and coordinate their processing. The components of this architecture, as applied to **Hiding Pot**, are shown in Figure 2.

---

1. The inferencing and frame selection needed to understand sentences such as **Hiding Pot** is explained more thoroughly in [Lange & Dyer, 1989a] and [Lange & Dyer, 1989b], which describe Robin, a model of high-level inferencing using an LCN without marker-passing.

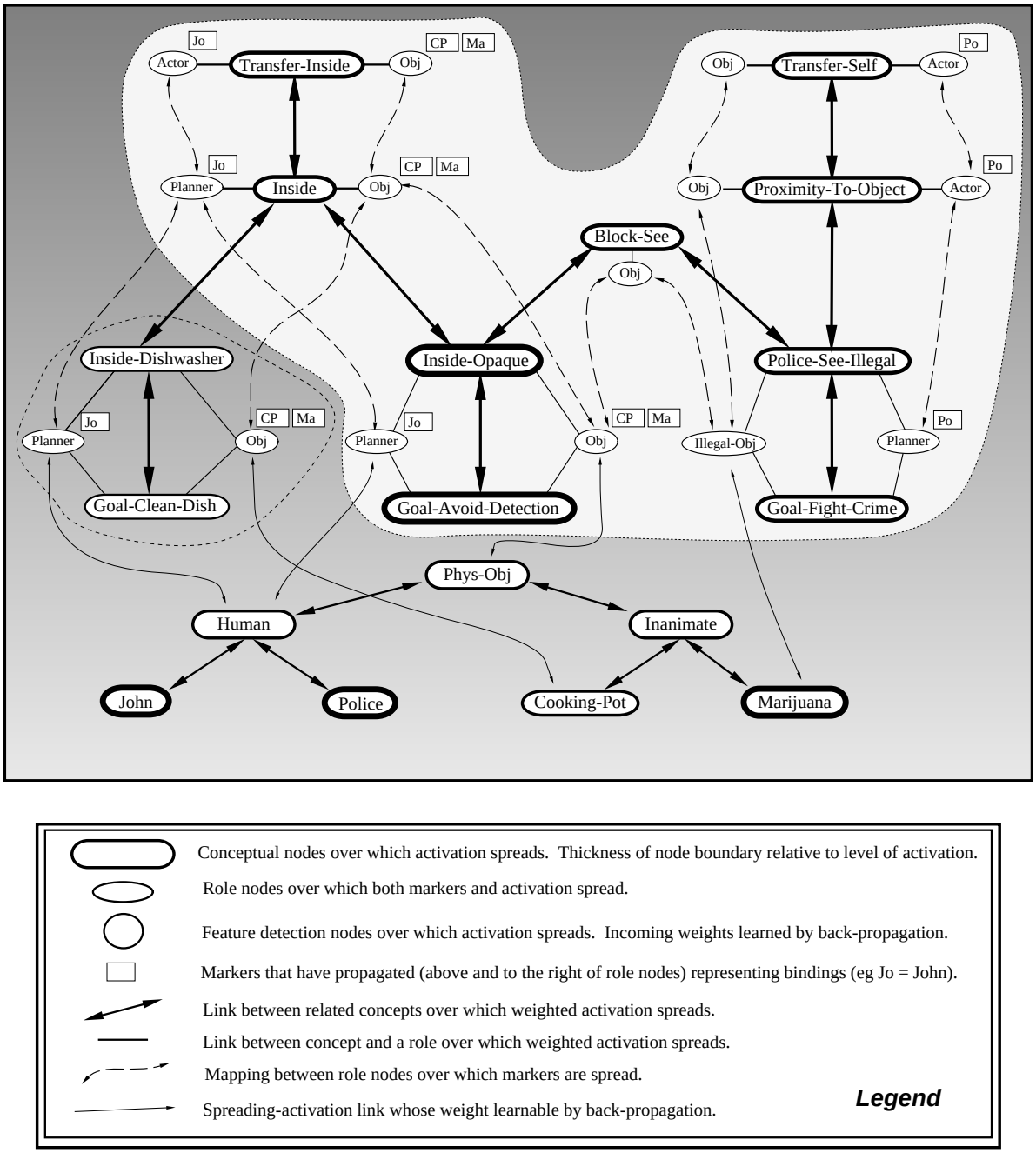

Figure 1: The sentence *"John put the pot in the dishwasher because the police were coming."* illustrates the utility of integrating semantic networks (Network-A) and distributed networks (Network-B). The darkest area represents the most highly-activated set of nodes representing the network's plan/goal analysis of the sentence. Not all markers are shown. Location role nodes and other parts of the network are also not displayed.

Processing Controllers

When Descartes is loaded and running, the required processing controllers are a *meta-controller* (a supervisor for all elements and sub-controllers present in the run-time system) and at least one *network controller* (a supervisor for an individual network and its elements). The architecture de-

scribed in Figure 2, and implemented in **Hiding Pot**, is controlled by a meta-controller (Meta-Control) which coordinates the two networks (Network-A and Network-B). Each of these networks has a local network controller which coordinates the processing of its elements. In this case the controller for Network-A is of class SA/MP-Control, which combines both spreading-activation and marker-passing functionality.

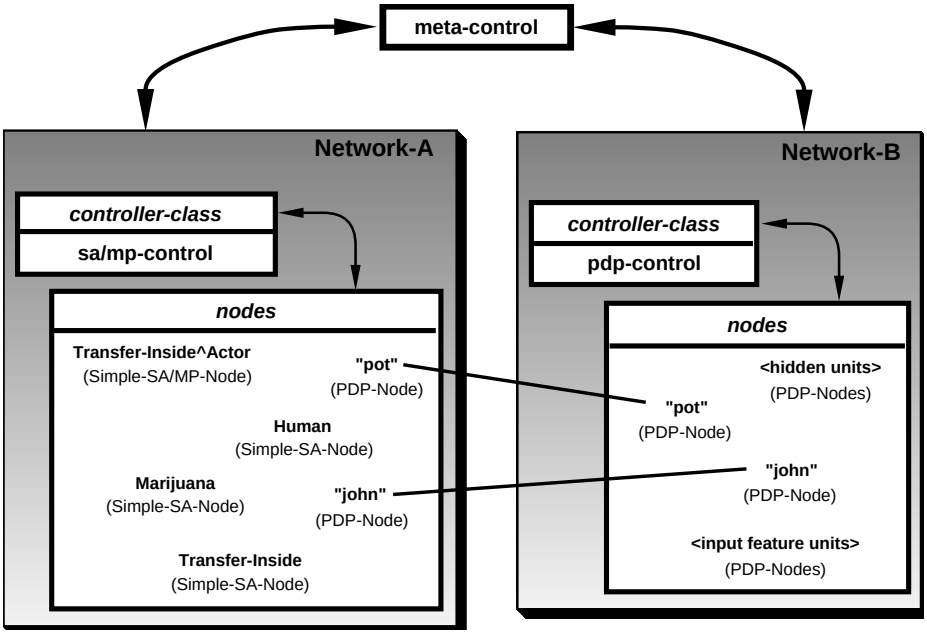

Figure 2: Descartes Processing Architecture applied to **Hiding Pot**. Shown in each network are a few of their nodes, with the class of each node being declared in parentheses below their names. PDP-Nodes "pot" and "john" are shared by both networks.

Network Elements

The nodes shown in **Hiding Pot** are illustrative of the kinds of nodes provided in the system. Three of Descartes's predefined node classes are used in **Hiding Pot**: (1) Simple-SA-Node, used in **Hiding Pot** for conceptual elements, such as Human and Transfer-Inside, (2) Simple-SA/MP-Node, used for roles, such as Transfer-Inside^Actor, and (3) PDP-Node, used for feature detection in Network-B, such as the node representing lexical entry "pot". Figure 3 provides an example of node creation in Descartes.

Simple-SA-Node is a basic class of spreading-activation nodes with default activation and output functions. Simple-SA/MP-Node is another standard node class, which combines the functionality of Simple-SA-Node with that required for marker passing. Finally, PDP-Node is the simplest class of DCN-type nodes — spreading-activation nodes that modify the weights on their input links by backpropagation [Rumelhart *et al.*, 1986, chap. 8].

Many other common node and link types are predefined, with a variety of activation, threshold, and output functions. More complicated classes are also available, including gated nodes and links, along with more neurally-realistic nodes that communicate via output spikes, such as the artificial neural oscillators of [Vidal & Haggerty, 1987]. The functionality of Descartes objects can easily be extended by combining the default class definitions of the object hierarchy with user-defined modifications, a process described in [Lange *et al.*, 1989].

```
(Simple-SA-Node Transfer-Inside :in-links (SA-Link ("put"                 0.75)
                                                   (Inside                1.00)
                                                   (Transfer-Inside^Actor 0.50)
                                                   (Transfer-Inside^Obj   0.50)
                                                  (Transfer-Inside^Loc   0.50)))

(Simple-SA/MP-Node Transfer-Inside^Actor :in-links (SA-Link (Transfer-Inside
1.0))
                                                   (MP-Link Inside^Planner))
```

Figure 3: Creation of Transfer-Inside and Transfer-Inside^Actor nodes, with forward-referencing.

Structured Networks

Some connectionist models have a consistent structure between groups of nodes in the network. In a semantic network, for example, a node representing the head of a frame might always be connected via a certain type of link to each of its roles, which in turn might always have a node for their fillers. Groups of nodes forming winner-take-all networks are always completely interconnected with constant inhibitory weights. Rather than force the user to repetitively define all nodes and connections for each such structured group, Descartes has a facility that allows the programmer to optionally define a structured growing method for each node class. A node's growth method automatically creates the node's expected structured incoming and outgoing nodes and connections. This feature allows knowledge base definitions to act as keys for network creation rather than as exhaustive listings of the networks' nodes and their connectivity.

Simulation in DESCARTES

Once the networks have been designed and built, the user starts the simulation by (1) optionally defining the cycling, termination, and display sequence for each network, (2) initializing the meta-controller to clear out all activation and markers, (3) activating or marking the desired nodes, and (4) starting the cycling sequence and specifying the number of global cycles to run. An example of this process is shown in Figure 4, but for a complete description see [Lange *et al.*, 1989].

Figure 4 shows the initial activation and markers needed to process the phrase *"John put the pot inside the dishwasher."* The first define-cycling command in the figure specifies that the meta-controller spread activation in Network-A once per global cycle, while only passing markers once per every three global cycles. Both activation and markers will cycle until stability, their default termination condition. For analysis of the network's activity, the user has defined that a trace of the markers' propagation be shown and that the status of the nodes be displayed every ten cycles. The second define-cycling command defines that Network-B is not to be cycled in this example.

In general, the networks' cycling sequences need only be set once per session (if at all), although all sequencing and displaying parameters may be re-specified in mid-simulation. Activations and markers of nodes may be changed at any time. The cycling sequence is further described below.

The Simulation Cycle

As shown, Descartes is designed in such a way that networks can be cycled in parallel or serially. The meta-controller provides for timing coordination between the networks. Networks cycled in parallel behave as if they were a single net, even though they need not operate at the same frequency, or, in fact, with the same functionality. A particular model may have a network of inhibitory nodes cycling at a faster rate than a network of excitatory nodes with which it interacts, at the same time as symbolic markers are being passed over each, and backpropagation is being performed within sub-networks of the model. With serial cycling, one network may wait until another network completes a specified number of cycles or reaches stability before starting to cycle itself.

```
(define-cycling %Network-A :sa-cycle-every     1    ;; (1)
                           :marker-cycle-every 3
                           :marker-trace       T
                           :display-every      10)
(define-cycling %Network-B :sa-cycle-every     NIL)
(init meta-control)                                 ;; (2)

(clamp-activation %"put"     1.0)                   ;; (3)
(mark %Transfer-Inside^Actor (marker %John))
(mark %Transfer-Inside^Obj   (marker %Cooking-Pot)
                             (marker %Marijuana))
(mark %Transfer-Inside^Loc   (marker %Dishwasher))

(cycle 50)                                          ;; (4)
```

Figure 4: An example of the Descartes control language.

Each global network cycle is comprised of four steps: (1) determination of which networks need to be cycled, (2) update of active nodes in the cycling networks, (3) spread from active nodes in the cycling networks to their out-links, and (4) report any requested output.

Determining Active Networks: The meta-controller determines which of the networks in the system need to be cycled in parallel on the given cycle, according to defaults and any define-cycling commands. In Figure 4, spreading-activation nodes in Network-A will be cycled on every global cycle, while marker-passing nodes will be cycled only on global cycles 1, 4, 7, and so on, until termination (stability).

Update: Each active node in the cycling networks queries its incoming links for new activation and/or markers. Spreading-activation nodes calculate their new activation by applying their activation function, while marker-passing nodes store any new markers they have received.

Spread-To-Out-Links: Each active node in the cycling networks calculates its output (either activation or markers) and sends it to its outgoing links. The output of spreading-activation nodes is calculated by applying their output function, while the output of marker-passing nodes is generally their new markers.

Report Output: The final step of a cycle entails querying the cycling networks for results. Each network controller can optionally display the status of important nodes at specified cycles (Network-A's status will be displayed every 10 cycles in Figure 4) or trace new activation and/or markers. Descartes currently has a number of output options useful for system design and debugging.

Implementation and Simulator Access

Descartes has been designed for portability, flexibility, and simplicity of use. Portability is achieved via the use of CommonLisp, the ANSI Lisp standard. Flexibility is augmented by the use of the CommonLisp Object System, Clos, whose hierarchical class structure provides inheritance which enables users to utilize pre-defined functional classes to customize their own semantics. A complete description of currently available functionality and test-bed cases can be found in [Lange *et al.*, 1989]. The largest test case simulated to date is an implementation of a Robin [Lange & Dyer, 1989b] network in the domain of **Hiding Pot**. It consists of two interacting LCNs built from four node classes and five link classes, with a total of 12,400 nodes and 40,000 links.

Descartes's control language is simple and effective, enabling the designer to easily set up and test different network configurations using either pre-defined or user-defined elements. At the same

time, the system has been designed with ease of network debugging in mind, with history and output facilities that offer researchers valuable methods for interpreting network behavior.

Descartes will be made available to all interested users. Enquiries about access to the simulator should be sent to DESCARTES@CS.UCLA.EDU.

Related Work

Some of the recent tools constructed for building and simulating connectionist architectures are (1) the Rochester Connectionist Simulator (RCS) [Goddard *et al.*, 1987], (2) the PDP Software Package [McClelland & Rumelhart, 1988], (3) Mirrors/II [D'Autrechy *et al.*, 1988], and (4) Genesis [Wilson *et al.*, 1988]. RCS is a spreading-activation simulator which allows units to have any amount of associated data. There is no specification language for construction of the net, but the system provides a library of commonly used network structures and units. The PDP Software package includes a number of programs for simulating the DCN models in [Rumelhart & McClelland, 1986]. Mirrors/II and Genesis, the most recent of the four systems, have both features: a high level non-procedural language for network construction and an indexed library of commonly used networks. Both have more sophisticated and flexible control mechanisms than RCS and the PDP Software Package, with Mirrors/II emphasizing simulations using LCNs and Genesis emphasizing realistic, biologically-based models.

The flexibility and symbolic capabilities afforded by Descartes' object-oriented implementation in CommonLisp and Clos comes at a small expense in simulation speed in comparison to the C-based implementations of RCS, the PDP package, and Genesis. The only case where the difference in speed should be significant, however, is in simple backpropagation networks requiring thousands of learning epochs, for which the PDP package might be more appropriate. Except for Genesis, all of the above-mentioned simulators are geared toward monotonic distributed or localist spreading-activation networks. None of them have the concept of hybrid multiple interactive networks as part of their design, especially those which can pass symbolic markers.

Conclusions

We have presented a development tool, DESCARTES, which provides researchers with the capability to combine Distributed Connectionist Networks, Localist Connectionist Networks and Marker-Passing Networks within a single simulation environment. The most important theoretical contribution of DESCARTES is the concept of Multiple Interactive Networks with intra- and inter-network heterogeneity. As a tool, it provides a simple, portable, and versatile environment for designing and testing different cognitive models. These capabilities make DESCARTES a unique and powerful tool for researchers in Artificial Intelligence, Cognitive Modelling, and Connectionism.

Acknowledgements

This research has been supported in part by a contract with the JTF program of the DOD and the Office of Naval Research (no. N00014-86-0615). DESCARTES has been implemented on equipment donated to UCLA by Hewlett-Packard, Inc., and Apollo Computer, Inc. We would like to thank John Reeves, Colin Allen, Michael Dyer, and Eduard Hoenkamp for their helpful comments on previous drafts of this paper.

References

D'Autrechy, C. L., Reggia, J. A., Sutton, G. G., & Goodall, S. M. (1988): A General-Purpose Simulation Environment For Developing Connectionist Models. *Simulation, 51(1),* p. 5-19.

Charniak, E. (1986): A Neat Theory of Marker Passing. *Proceedings of the National Conference on Artificial Intelligence (AAAI-86)*, Philadelphia, PA, 1986.

Dyer, M. G. (1989): Symbolic Neuroengineering for Natural Language Processing: A Multi-Level Research Approach. In J. Barnden and J. Pollack, editors, *Advances in Connectionist and Neural Computation Theory*, Ablex Publishing, 1989. In press.

Goddard, N., Lynne, K. J., & Mintz, T. (1986): *Rochestor Connectionist Simulator*. Technical Report TR-233, Department of Computer Science, University of Rochestor.

Hendler, J. (1988): *Integrating Marker-Passing and Problem Solving: A Spreading Activation Approach to Improved Choice in Planning*, Lawrence Erlbaum Associates, Hillsdale, New Jersey.

McClelland, J. L. & Rumelhart, D. E. (1988): *Explorations in Parallel Distributed Processing: A Handbook of Models, Programs, and Exercises*, MIT Press, Cambridge, MA.

Lange, T. & Dyer, M. G. (1989a): Dynamic, Non-Local Role-Bindings and Inferencing in a Localist Network for Natural Language Understanding. In David S. Touretzky, editor, *Advances in Neural Information Processing Systems I*, p. 545-552, Morgan Kaufmann, San Mateo, CA (Collected papers of the IEEE Conference on Neural Information Processing Systems — Natural and Synthetic, Denver, CO, November 1988).

Lange, T. & Dyer, M. G. (1989b): Frame Selection in a Connectionist Model of High-Level Inferencing. *Proceedings of the Eleventh Annual Conference of the Cognitive Science Society (CogSci-89)*, Ann Arbor, MI, August 1989.

Lange, T., Hodges J. B., Fuenmayor, M., & Belyaev, L. (1989): *The DESCARTES Users Manual.* Research Report, Computer Science Department, University of California, Los Angeles.

Nenov, V. I., & Dyer, M. G. (1988): DETE: A Connectionist/Symbolic Model of Visual and Verbal Association. *Proceedings of the IEEE Second Annual International Conference on Neural Networks (ICNN-88)*, San Diego, CA, July 1988.

Rumelhart, D. E., & McClelland, J. L. (1986): *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*. Volumes 1-2, MIT Press, Cambridge, MA.

Shastri, L. (1988): A Connectionist Approach to Knowledge Representation and Limited Inference. *Cognitive Science, 12*, p. 331-392.

Vidal, J. & Haggerty, J. (1987): Synchronization in Neural Nets. *Proceedings of the IEEE Conference on Neural Information Processing Systems — Natural and Synthetic (NIPS-87)*, Denver, CO, November 1987.

Waltz, D. & Pollack, J. (1985): Massively Parallel Parsing: A Strongly Interactive Model of Natural Language Interpretation. *Cognitive Science, 9(1)*, p. 51-74.

Wilson, M. A., Upinder, S. B., Uhley, J.D., & Bower, J. M. (1988): GENESIS: A System for Simulating Neural Networks. In David S. Touretzky, editor, *Advances in Neural Information Processing Systems I*, p. 485-492, Morgan Kaufmann, San Mateo, CA (Collected papers of the IEEE Conference on Neural Information Processing Systems — Natural and Synthetic, Denver, CO, November 1988).