To appear in the *Proceedings of the International Symposium on Integrating Knowledge and Neural Heuristics (ISIKNH'94)*; Pensacola, FL; May 1994.

Genetically Refining Topologies of Knowledge-Based Neural Networks*

David W. Opitz and **Jude W. Shavlik**
Computer Sciences Department
University of Wisconsin – Madison
Madison, WI 53706, U.S.A.
{opitz,shavlik}@cs.wisc.edu

Abstract

Traditional approaches to connectionist theory refinement map the dependencies of a domain-specific rulebase into a neural network, then refine these reformulated rules using neural learning. These approaches have proven to be effective at classifying previously unseen examples; however, most of these approaches suffer in that they are unable to refine the topology of the networks they produce. Thus, when given an *impoverished* domain theory, they generalize poorly. A recently published improvement to these approaches, the TopGen algorithm, addressed this limitation by heuristically searching expansions to the knowledge-based networks produced by these algorithms. We show, however, that TopGen's search is too restricted. In response, we present the Regent algorithm, which uses genetic algorithms to broaden the type of networks seen during its search. It does this by using (a) the domain theory to help create an initial population and (b) crossover and mutation operators specifically designed for knowledge-based networks. Experiments on three real-world domains indicate that our new algorithm is able to significantly increase generalization when compared to both TopGen and a standard approach that does not alter its knowledge-based network's topology.

1 Introduction

Inductive learning systems that utilize a set of approximately correct, domain-specific inference rules (called a *domain theory*), describing what is currently known about the domain, are called *theory-refinement* systems. Being able to make use of a domain theory is desirable because inductive learners that start with an approximately correct theory can

*This work was supported by DOE grant DE-FG02-91ER61129, ONR grant N00014-93-1-0998, and NSF grant IRI-9002413.

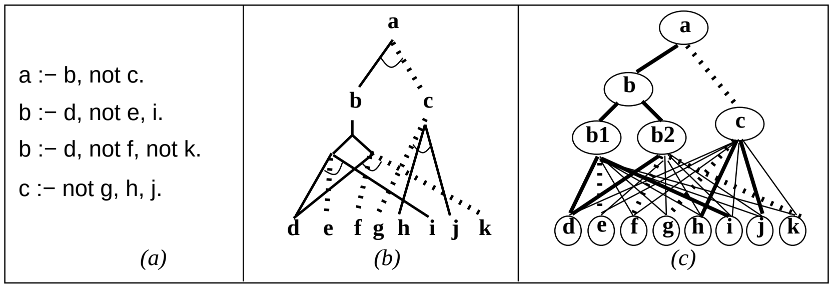
achieve high generalization[1] with significantly fewer examples (Ourston & Mooney, 1990; Towell et al., 1990; Pazzani & Kibler, 1992). Several theory-refinement systems use neural networks as their inductive learning component. These knowledge-based connectionist approaches have been shown to frequently generalize better than many other machine learning systems (Fu, 1989; Towell, 1991; Tresp et al., 1992; Lacher et al., 1992; Opitz & Shavlik, 1993). In this paper, we present such an approach, called REGENT (REfining, with Genetic Evolution, Network Topologies), that uses genetic algorithms, along with the aid of a domain theory, to search for a good neural network topology.

KBANN (Towell et al., 1990) is an example of a connectionist theory-refinement system that translates the provided domain theory into a neural network, thereby determining the network's topology. It then refines these reformulated rules using backpropagation. However, KBANN, and other connectionist theory-refinement systems that do not alter their network topologies, suffer when given *impoverished* domain theories – ones that are missing rules needed to adequately learn the true concept (Towell & Shavlik, 1992; Opitz & Shavlik, 1993). TopGen (Opitz & Shavlik, 1993) is an improvement to these systems; it heuristically searches through the space of possible network topologies by adding hidden nodes to the neural representation of the domain theory. TopGen showed statistically significant improvements over KBANN in several real-world domains (Opitz & Shavlik, 1993); however, in this paper we empirically show that as we increase the number of networks considered, TopGen suffers because it only considers simple expansions of the KBANN network. Being able to effectively use all available computing power to search many candidate networks is desirable because (a) computing power is rapidly growing and (b) for many applications, it is more important to obtain concepts that generalize well than it is to induce concepts quickly.

To address TopGen's limitation, we broaden the type of topologies that TopGen considers by using genetic algorithms (GAs). GAs have been shown to be effective optimization techniques because of their efficient use of global information (Holland, 1975; Goldberg, 1989). Our new algorithm, REGENT, proceeds by first trying to generate, from the domain theory, a diversified initial population. It then produces new candidate networks via the genetic operators of *crossover* and *mutation*, which we tailored for knowledge-based neural networks. REGENT's crossover operator tries to maintain the rule structure of the network, while its mutation operator adds nodes to a network by using the TopGen algorithm. Hence, REGENT mainly differs from other techniques that use GAs to determine a network topology (Miller et al., 1989; Dodd, 1990; Harp et al., 1991; Romaniuk, 1993) in that its genetic operators are specialized for connectionist theory refinement. Experiments reported herein show that REGENT is able to better search for network topologies than is TopGen.

The rest of the paper is organized as follows. In the next section, we give a brief review of the KBANN and TopGen algorithms. We present the details of the REGENT algorithm in Section 3. This is followed by results from three real-world Human Genome domains. In Section 5, we discuss these results, as well as give future work, before concluding.

[1] We use *generalization* to mean accuracy on examples not seen during training.



a :- b, not c.
b :- d, not e, i.
b :- d, not f, not k.
c :- not g, h, j.

Figure 1: Translation of a knowledge base into a neural network.

2 The KBANN and TopGen Algorithms

We start our search for a good network topology by choosing, as an initial guess, the network defined by the KBANN algorithm. Figure 1 illustrates how KBANN translates a set of propositional rules, representing what is initially known about a domain, into a neural network. Figure 1a shows a Prolog-like rule set that defines membership in category a. Figure 1b represents the hierarchical structure of these rules, with solid lines representing necessary dependencies and dotted lines representing prohibitory dependencies. Figure 1c represents the resulting network created from this translation. KBANN creates nodes $b1$ and $b2$ in Figure 1c to handle the two rules defining b in the rule set. Biases are set to represent the appropriate AND or OR structure of each corresponding node. The thin lines in Figure 1c are lightly-weighted links that KBANN adds to allow refinement of these rules during backpropagation training. This training alters the antecedents of existing rules; however, KBANN does not have the capability of inducing new rules. For example, KBANN is unable to add a third rule for inferring b. Thus KBANN suffers when given domain theories that are missing rules needed to adequately learn the true concept (Towell & Shavlik, 1992; Opitz & Shavlik, 1993).

TopGen addresses this limitation by heuristically searching through the space of possible expansions to the KBANN network. TopGen proceeds by first training the KBANN network, then placing it on a search queue. In each cycle, TopGen takes the best network from the search queue, decides where and how to add new nodes, trains these new networks, then places them back on the queue. TopGen judges where errors are in the network by using the training examples to increment two counters for each node, one for false negatives and one for false positives. It then adds nodes to the network in a manner analogous to adding rules and conjuncts to a symbolic rule base. Adding nodes in this fashion helps to correct the types of errors that KBANN is ineffective at correcting. For example, KBANN is effective at removing antecedents from existing rules (Towell, 1991), so TopGen attempts to decrease false negatives by adding nodes in a fashion analogous to adding a new rule to the rule base. TopGen showed statistically significant improvements over KBANN in several real-world domains and comparative experiments with a simple approach to adding nodes verified that new nodes must be added in an intelligent manner (Opitz & Shavlik, 1993).

Despite this success, TopGen suffers in that it only considers larger networks that contain

Table 1: The REGENT Algorithm.

GOAL: Search for the best network topology describing the domain theory and data.

1. Set aside a validation set from the training instances.
2. Perturb the KBANN-produced network in multiples ways to create initial networks, then train these networks and place them into the population.
3. Loop forever:
   (a) Create new networks using the crossover or mutation operator.
   (b) Train these networks with backpropagation, score with the validation set, and place into the population.
   (c) If a new network is the smallest network with the lowest validation-set error seen so far, report it as the current best concept.

---

the original KBANN network. In this paper, we increase the number of networks TopGen considers during its search and show that its increase in generalization is primarily limited to the first few networks searched. Thus when TopGen has time to consider many candidate networks, it is unable to effectively utilize all of this time to efficiently explore topology space. *Broadening the range of networks considered during the search through topology space is the major focus of this paper.*

3 The REGENT Algorithm

Our new algorithm, REGENT, tries to broaden the types of networks considered with the use of GAs. We view REGENT as having two phases: (1) genetically searching through topology space, and (2) training each network using backpropagation. REGENT utilizes the domain theory to aid in both phases. It uses the theory to help guide its search through topology space and to give a good starting point in weight space.

Table 1 summarizes the REGENT algorithm. REGENT first sets aside a *validation* set (from part of the *training* instances) for use in scoring the different networks. It then perturbs the KBANN-produced network to create an initial set of candidate networks. Next, REGENT trains these networks using backpropagation and places them into the population. In each cycle, REGENT creates new networks by crossing over and mutating networks from the current population that are randomly picked proportional to their fitness (i.e., validation-set correctness). It then trains these new networks, and places them into the population. As it searches, REGENT keeps the network that has the lowest validation-set error as the best concept seen so far, breaking ties by choosing the smaller network in an application of Occam's Razor.

A diverse initial population will help to broaden the types of networks REGENT considers during its search; however, we still need to utilize the domain theory when generating this population. REGENT does this by randomly perturbing the KBANN network at various

Table 2: REGENT's method for crossing over two networks.

Crossover Two Networks:
GOAL: Crossover two networks to generate two new network topologies.

1. Divide each network's hidden nodes into sets `A` and `B` using **DivideNodes**.
2. From the two sets `A` and `B`, form new networks as follows:
   (a) Keep links between nodes coming from the same network.
   (b) Link unconnected nodes between levels with near-zero weights.
   (c) Adjust node biases to keep original AND or OR function of each node.

DivideNodes:
GOAL: Divide the hidden nodes into sets `A` and `B`, while retaining each network's rule structure.

```
While some hidden node is not assigned to set A or set B:
    (i) Collect the unassigned hidden nodes whose output is linked only to either previously-
        assigned nodes or outputs nodes.
    (ii) If set A or set B is empty:
            For each node collected in part (i), randomly assign it to set A or set B.
         Else
            Probabilistically add the nodes collected in part (i) to set A or set B. Equation (1)
            shows the probability of being assigned to set A. The probability for being assigned
            to set B is one minus this value.
```

---

nodes. A node is perturbed by either deleting it, or by adding new nodes to it in a manner analogous to one of TopGen's four methods for adding nodes. If there are multiple theories about a domain, all of them can be used to seed the population.

REGENT crosses over two networks by first dividing the nodes in each parent network into two sets, `A` and `B`, then combining the nodes in each set to form two new networks (i.e., the nodes in the two `A` sets form one network, while the nodes in the two `B` sets form another). Table 2 summarizes REGENT's method for crossover and Figure 2 gives an example. REGENT divides nodes, one level at a time, starting at the level nearest the output nodes. When considering a level, if either set `A` or set `B` is empty, it cycles through each node in that level and randomly assigns it to either set. If neither set is empty, nodes are probabilistically placed into a set. The following equation calculates the probability of a given node being assigned to set `A`:

$$Prob(node\ i\ assigned\ to\ set\ A) = \frac{\sum_{j\in A}|w_{ji}|}{\sum_{j\in A}|w_{ji}| + \sum_{j\in B}|w_{ji}|} \qquad (1)$$

where $j \in A$ means node j is a member of set `A` and w_{ji} is the weight value from node i to node j. The probability of belonging to set `B` is one minus this probability. With these probabilities, REGENT tends to assign nodes that are heavily-linked together to the same set. This helps keep intact the rule structure of the crossed-over networks. When creating the links in the new networks, REGENT first retains the links connecting two nodes that

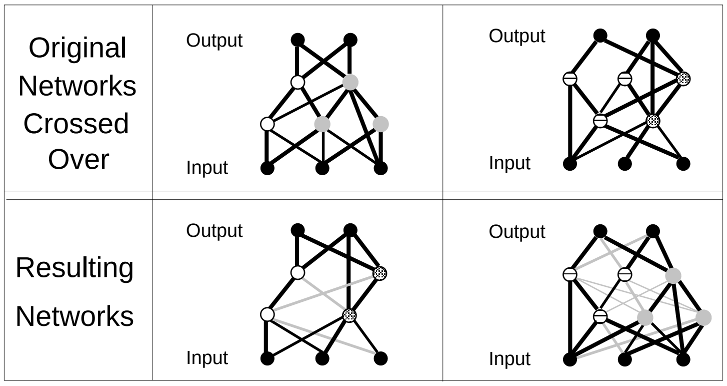


Figure 2: REGENT's method for crossing over two networks. The hidden nodes in each original network are divided into the sets `A` and `B`; the nodes in the two `A` sets form one new network, while the nodes in the two `B` sets form another new network. Grey lines represent low-weighted links that are added to fully-connect neighboring levels.

come from the same original network. It then adds low-weighted links between unconnected nodes on consecutive levels.[2] Finally, it adjusts the bias of the nodes to maintain their AND or OR function.[3]

REGENT mutates networks by applying a variant of TopGen. REGENT uses TopGen's method for incrementing the false-negatives and false-positives counters for each node. REGENT then adds nodes, based on the values of these counters, the same way TopGen does. This mutation operator adds diversity to a population, while still benefitting from a directed, heuristic-search technique for choosing where to add nodes.

REGENT adds newly trained networks to the population only if their validation-set correctness is better than or equal to an existing member of the population. When REGENT replaces a member, it chooses the oldest member having the lowest correctness. Other techniques (Goldberg, 1989), such as replacing the member nearest the new candidate network, can promote diverse populations; however, we do not want to promote diversity at the expense of decreased generalization. Therefore, we currently do not use these techniques since we are not yet able to consider thousands of networks and thus have not had trouble with converging too quickly. Once we are able to consider many more networks, we plan to investigate incorporating diversity-promoting techniques.

4 Experimental Results

We ran REGENT on three problems from the Human Genome Project. Each of these problems aid in locating genes in DNA sequences. The first domain, *promoter recognition*, con-

[2]A node's *level* is defined as the longest path from it to an output node.

[3]If a positive incoming link for an AND node is removed, the node's bias is decremented by subtracting the product of the link's magnitude times the average activation entering that link. The bias for an OR node is incremented by a similar amount when negative incoming links are removed.

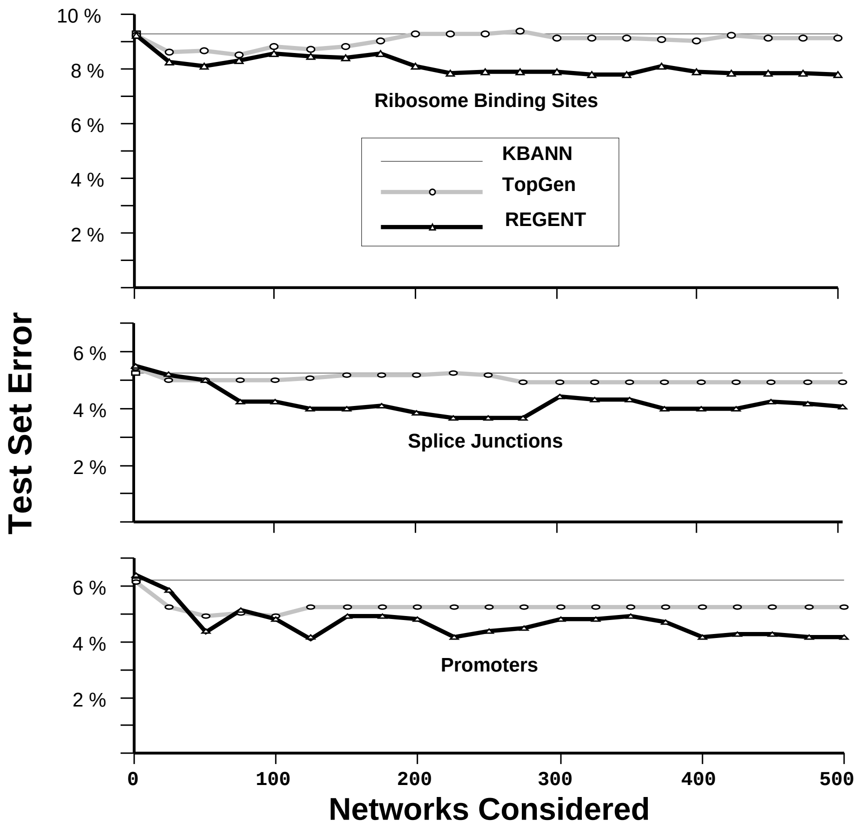


Figure 3: Error rates on three Human Genome problems.

tains 234 positive examples, 702 negative examples, and 17 rules. The second domain, *splice-junction determination*, contains 1,200 examples distributed equally among three classes, and 23 rules. Finally, the *ribosome binding sites* (RBS) domain, contains 366 positive examples, 1,098 negative examples, and 17 rules. (Note that these data sets and domain theories are different versions of the ones that appear in Towell et al., 1990, and Opitz & Shavlik, 1993.)

Our experiments address the test-set accuracy of Regent on these domains. Figure 3 shows the test-set error of Kbann, TopGen, and Regent as they search through the space of network topologies. The results are from a ten-fold cross validation; in each fold, Regent is run with a population size of 20. The horizontal line in each graph results from the Kbann algorithm. Even though Kbann considers only one network, we drew a horizontal line for the sake of visual comparison. The first point of each graph, after one network is considered, is nearly the same for all three algorithms. This occurs because all three algorithms start with the Kbann network; however, TopGen and Regent do not train the network with all of the training data, since they hold some aside for a validation set. Notice that TopGen stops improving after considering 10 to 30 networks and that the generalization ability of Regent is better than TopGen after this point.

Figure 4 shows the test-set error after TopGen and Regent consider 500 candidate topologies. The standard neural network results are from a fully-connected, single-layer feed-forward neural network, where, for each fold, we trained various networks containing up

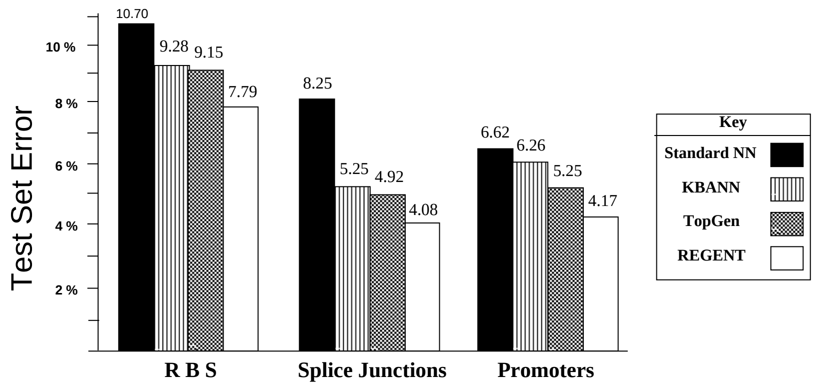


Figure 4: Error rates after TopGen and REGENT consider 500 networks. Two-sample one-tailed *t*-tests indicate that REGENT differs from both KBANN and TopGen at the 92.5% confidence level on all three domains.

to 100 hidden nodes and used a validation set to choose the best network. Our results show KBANN generalizes much better than the best of these standard networks, thus confirming KBANN's effectiveness in generating good network topologies. While TopGen is able to improve on the KBANN network, REGENT is able to significantly decrease the error rate over both KBANN and TopGen.

5 Discussion and Future Work

Towell (1992) has shown that KBANN generalizes better than many other machine learning algorithms, including purely symbolic approaches to theory refinement, on the promoter and splice-junction domains (the RBS dataset did not exist then). Despite this success, REGENT is able to significantly improve generalization over both KBANN and an improvement to KBANN, the TopGen algorithm. REGENT reduces KBANN's test-set error by 16% for the RBS domain, 22% for the splice junction domain, and 33% for the promoter domain; it reduces TopGen's test-set error by 15% for the RBS domain, 17% for the splice junction domain, and 21% for the promoter domain. Also, REGENT's ability to utilize available computing time is further aided by its being inherently parallel, since we can train many networks simultaneously.

Since we are searching through many candidate networks, it is important to be able to recognize the networks that are likely to generalize the best. We currently use a validation set; however, MacKay (1992) has shown that a validation set can be a noisy estimate of the true error. Also, as we increase the number of networks searched, REGENT may start selecting networks that overfit the validation set. Future work, then, is to investigate selection methods, such as Bayesian techniques (MacKay, 1992), that do not use a validation set. This would also allow us to use all the training instances to train the networks.

Also, since the correct theory may be far from the initial domain theory, we plan to evaluate including, in the initial population of networks, a variety of networks not obtained directly from the domain theory. Currently, we create our initial population by always perturbing the original KBANN network using TopGen's four methods for adding nodes. To include networks that are not obtained from the domain theory, we plan to use TopGen's node-addition techniques to randomly create all of the hidden nodes in a network. Adding nodes in this manner creates networks whose node structure is analogous to dependencies found in symbolic rule-bases, thus creating networks designed for REGENT's crossover and mutation operators.

Finally, since REGENT considers many networks, it can select a subset of the final population of networks and then use a collective decision strategy at minimal extra cost. Hansen and Salamon (1990) showed that combining the output of several neural networks will improve generalization if the individual networks tend to be independent in their error. To help promote this independence, we plan to investigate incorporating techniques that help create subpopulations (Goldberg, 1989), then select a network from each subpopulation.

6 Conclusion

Connectionist theory-refinement systems have been shown to be effective at translating a domain theory into a neural network; however, most of these systems, such as the KBANN algorithm, suffer in that they do not alter their topology. TopGen is an improvement to KBANN that uses available computer power to search for effective places to add nodes to the KBANN network; however, we showed empirically that TopGen suffers from restricting its search to expansions of the KBANN network, and is unable to improve its performance after searching beyond a few topologies. Therefore TopGen is unable to exploit all available computing power to increase the correctness of an induced concept.

We presented a new algorithm, REGENT, that uses genetic algorithms to broaden the types of topologies considered during TopGen's search. Experiments indicate that REGENT is able to significantly increase generalization over TopGen; hence, our new algorithm is successful in overcoming TopGen's limitation of only searching a small portion of the space of possible network topologies. In doing to, REGENT is able to generate a good solution quickly, by using KBANN, then is able to continually improve this solution as it searches concept space. Thus our new algorithm further increases the applicability of connectionist learning to problems containing preexisting, domain-specific knowledge.

References

Dodd, N. (1990). Optimization of network structure using genetic techniques. In *Proceedings of the IEEE International Joint Conference on Neural Networks (volume III)*, (pp. 965–970), Paris.

Fu, L. M. (1989). Integration of neural heuristics into knowledge-based inference. *Connection Science*, 1:325–340.

Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison-Wesley, Reading, MA.

Hansen, L. K. & Salamon, P. (1990). Neural network ensembles. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12:993–1001.

Harp, S. A., Samad, T., & Guha, A. (1991). Designing application-specific neural networks using the genetic algorithm. In Touretzky, D., editor, *Advances in Neural Information Processing Systems (volume 2)*, (pp. 447–454), San Mateo, CA. Morgan Kaufmann.

Holland, J. H. (1975). *Adaptation in Natural and Artificial Systems.* University of Michigan Press, Ann Arbor, MI.

Lacher, R. C., Hruska, S. I., & Kuncicky, D. C. (1992). Back-propagation learning in expert networks. *IEEE Transactions on Neural Networks*, 3(1):62–72.

MacKay, D. J. (1992). A practical Bayesian framework for backpropagation networks. *Neural Computation*, 4:448–472.

Miller, G. F., Todd, P. M., & Hegde, S. U. (1989). Designing neural networks using genetic algorithms. In *Proceedings of the Third International Conference on Genetic Algorithms*, (pp. 379–384), San Mateo, CA. Morgan Kaufmann.

Opitz, D. W. & Shavlik, J. W. (1993). Heuristically expanding knowledge-based neural networks. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence*, (pp. 1360–1365), Chambery, France.

Ourston, D. & Mooney, R. J. (1990). Changing the rules: A comprehensive approach to theory refinement. In *Proceedings of the Eighth National Conference on Artificial Intelligence*, (pp. 815–820), Boston, MA.

Pazzani, M. & Kibler, D. (1992). The utility of knowledge in inductive learning. *Machine Learning*, 9:57–94.

Romaniuk, S. G. (1993). Evolutionary growth perceptrons. In *Proceedings of the Fifth International Conference on Genetic Algorithms*, (pp. 334–341), San Mateo, CA. Morgan Kaufmann.

Towell, G. & Shavlik, J. (1992). Using symbolic learning to improve knowledge-based neural networks. In *Proceedings of the Tenth National Conference on Artificial Intelligence*, (pp. 177–182), San Jose, CA.

Towell, G., Shavlik, J., & Noordewier, M. (1990). Refinement of approximate domain theories by knowledge-based neural networks. In *Proceedings of the Eighth National Conference on Artificial Intelligence*, (pp. 861–866), Boston, MA. AAAI Press.

Towell, G. G. (1991). *Symbolic Knowledge and Neural Networks: Insertion, Refinement, and Extraction.* PhD thesis, Computer Sciences Department, University of Wisconsin, Madison, WI.

Tresp, V., Hollatz, J., & Ahmad, S. (1992). Network structuring and training using rule-based knowledge. In Moody, J., Hanson, S., & Lippmann, R., editors, *Advances in Neural Information Processing Systems (volume 5)*, (pp. 871–878), San Mateo, CA. Morgan Kaufmann.