Appears in *Machine Learning: Proceedings of the Eleventh International Conference*,
W. Cohen & H. Hirsh, eds, Morgan Kaufmann, San Francisco, CA, 1994

# Using Genetic Search to Refine Knowledge-Based Neural Networks

**David W. Opitz and Jude W. Shavlik**
1210 W. Dayton St.
Computer Sciences Department
University of Wisconsin – Madison
Madison, WI 53706
{opitz, shavlik}@cs.wisc.edu

## Abstract

An ideal inductive-learning algorithm should exploit all available resources, such as computing power and domain-specific knowledge, to improve its ability to generalize. Connectionist theory-refinement systems have proven to be effective at utilizing domain-specific knowledge; however, most are unable to exploit available computing power. This weakness occurs because they lack the ability to refine the topology of the networks they produce, thereby limiting generalization, especially when given impoverished domain theories. 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 compared to a standard connectionist theory-refinement system, as well as our previous algorithm for growing knowledge-based networks.

## 1 INTRODUCTION

The task of inductive learning is to infer a concept given a set of training examples. An ideal inductive-learning algorithm should exploit all available resources, such as computing power and domain-specific knowledge, to improve its ability to generalize. Using domain-specific knowledge is desirable because inductive learners that start with an approximately correct theory can achieve high generalization[1] with significantly fewer examples (Ginsberg, 1990; Pazzani & Kibler, 1992; Ourston & Mooney, 1994; Towell & Shavlik, in press). Effectively using all available computing power is desirable because, for many applications, it is more important to obtain concepts that generalize well than it is to induce concepts quickly. In this paper, we present an algorithm, called REGENT (REfining, with Genetic Evolution, Network Topologies), that utilizes available computer time to extensively search for a neural-network topology that best explains the training data while minimizing changes to a domain-specific theory.

Inductive learning systems that utilize a set of approximately-correct, domain-specific inference rules (called a *domain theory*), which describe what is currently known about the domain, are called *theory-refinement* systems. For most domains, an expert who created the theory is willing to wait for weeks, or even months, if a learning system can produce an improved theory. Thus, given the rapid growth in computing power, we believe it is important to develop techniques that tradeoff the expense of large numbers of computing cycles for gains in predictive accuracy. Analogous to *anytime planning* techniques (Dean & Boddy, 1988), we believe machine learning researchers should create *anytime learning* algorithms.[2] Such learning algorithms should produce a good concept quickly, then continue to search concept space, reporting the new "best" concept whenever one is found.

We concentrate on connectionist theory-refinement systems, since they have been shown to frequently generalize better than many other inductive-learning and theory-refinement systems (Fu, 1989; Towell, 1991; Lacher et al., 1992; Tresp et al., 1992; Opitz & Shavlik, 1993). KBANN (Towell & Shavlik, in press) is an example of such a connectionist system; it translates the provided domain theory into a neural network, thereby determining the network's topology, and then refines the reformulated rules using backpropagation.

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

[2] Our use of the term *anytime learning* differs from Grefenstette & Ramsey (1992); they use it to mean continuous learning in a changing environment.

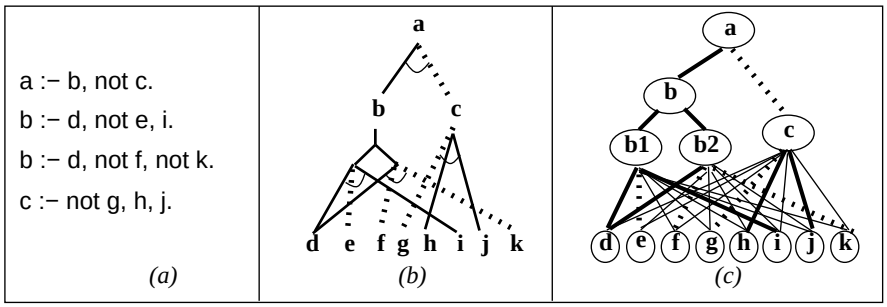


Figure 1: Translation of a knowledge base into a neural network. See Section 2 for details.

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 (Opitz & Shavlik, 1993; Towell & Shavlik, in press). TopGen (Opitz & Shavlik, 1993) is an improvement over 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 TopGen suffers because it only considers simple expansions of the KBANN network.

To address this 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; Koza, 1992). Our 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*. 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, our 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 briefly review the KBANN and TopGen algorithms. We present the details of our REGENT algorithm in Section 3. This is followed by results from three Human Genome Project domains. In Section 5, we discuss these results, as well as future work. We then review related work, before concluding.

## 2 REVIEW OF KBANN & TOPGEN

The goal of this research is to exploit both prior knowledge and available computing cycles to search for the neural network that is most likely to generalize the best to future examples. We proceed by choosing, as an initial guess, the network defined by the KBANN algorithm. KBANN translates a set of propositional rules, representing what is initially known about a domain, into a neural network.

Figure 1 illustrates this translation process. 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 (Opitz & Shavlik, 1993; Towell & Shavlik, in press).

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, estimates where errors are in the network, adds new nodes in response to these estimates, trains these new networks, then places them back on the queue. TopGen judges where errors are in a network by using training examples to increment two counters for each node, one for false negatives and one for false positives.

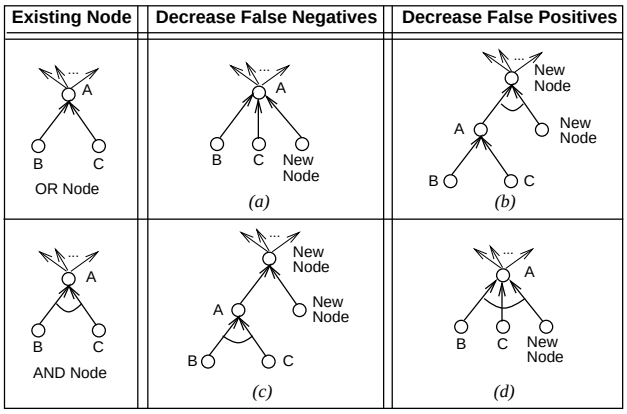


Figure 2: How TopGen adds new nodes to knowledge-based networks. Arcs indicate AND nodes.

Figure 2 shows the possible ways TopGen adds nodes to a TopGen network, based on these counter values. In a symbolic rulebase that uses negation-by-failure, we can decrease false negatives by either dropping antecedents from existing rules or adding new rules to the rulebase. Since KBANN is effective at removing antecedents from existing rules, TopGen adds nodes, intended to decrease false negatives, in a fashion that is analogous to adding a new rule to the rulebase (see Figure 2a,c). REGENT decreases false positives by constructively-inducing new antecedents (see Figure 2b,d). In doing so, TopGen is able to add rules, whose consequents were previously undefined, to the rulebase (something KBANN is incapable of doing).

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 the original KBANN network as subgraphs. 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 considered. 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: 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 multiple 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.

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 parallel version trains many candidate networks at the same time using the Condor system (Litzkow et al., 1988), which runs jobs on idle workstations.

A diverse initial population will 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 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

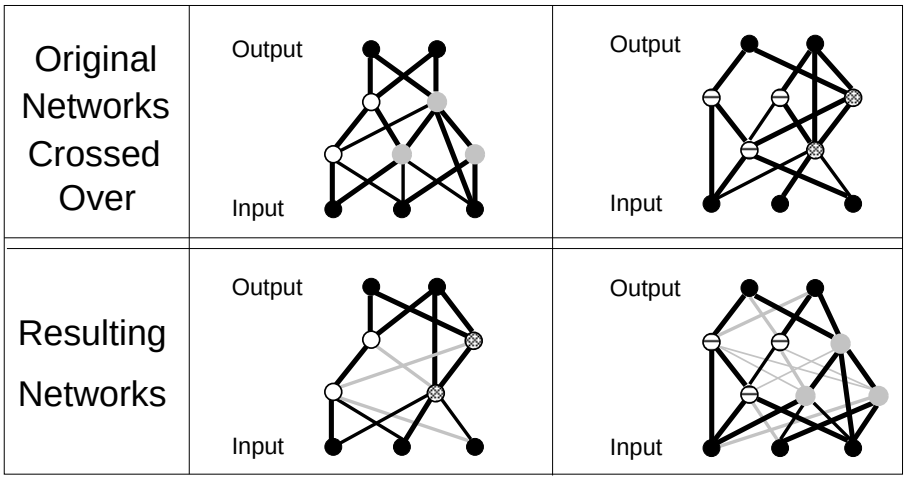


Figure 3: REGENT's method for crossing over two networks. The hidden nodes in each original network are divided into the sets A and B; the nodes in the two A sets form one new network, while the nodes in the two B sets form another. Grey lines represent low-weighted links that are added to fully-connect neighboring levels.

Table 2: REGENT's method for crossing over 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. Set A forms one network, while set B forms another. Each new network is created as follows:
   (a) A network inherits weight $w_{ji}$ from its parent if nodes $i$ and $j$ either are also inherited or are input or output nodes.
   (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 keeping each network's rule structure.

While some hidden node is not assigned to set A or 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 of being assigned to set B is one minus this value.

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 3 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:

$$Prob(node\ i \in setA) = \frac{\sum_{j \in A} |w_{ji}|}{\sum_{j \in A} |w_{ji}| + \sum_{j \in B} |w_{ji}|} \quad (1)$$

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 to the same set those nodes that are heavily-linked together. This helps to minimize the destruction of the rule structure of the crossed-over networks, since nodes belonging to the same syntactic rule are connected by heavily-linked weights. Thus, REGENT's crossover operator produces new networks by crossing-over rules, rather than just crossing-over nodes.

REGENT must then decide how to connect the nodes of the newly created networks. First, a new network inherits all weight values from its parents that connect two nodes that either it inherited, or are input or output nodes. It then adds low-weighted links between unconnected nodes on consecutive levels.[3] Finally, it

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

adjusts the bias of all AND or OR nodes to help maintain their *original* function.[4]

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 maintaining a directed, heuristic-search technique for choosing where to add nodes; this directedness is important because we currently are unable to consider more than a few thousand possible networks per day.

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 member having the lowest correctness (ties are broken by choosing the oldest member). 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. Once we are able to consider tens of thousands of networks, we plan to investigate incorporating diversity-promoting techniques.

REGENT can be considered a Lamarckian[5], genetic-hillclimbing algorithm (Ackley, 1987), since it performs local optimizations on individuals, then passes the successful optimizations on to offspring. Lamarckian learning can lead to a large increase in learning speed and solution quality (Farmer & Belin, 1992; Ackley & Littman, 1994).

## 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*, contains 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, 1991, and Opitz & Shavlik, 1993.)

Our experiments address the test-set accuracy of REGENT on these domains. Figure 4 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, since they all start with the KBANN network; however, TopGen and REGENT differ slightly from KBANN since they must set aside part of the training set to score the candidate networks. 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 5 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 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 & FUTURE WORK

Towell (1991) 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 12% for the RBS domain, 22% for the splice-junction domain, and 33% for the promoter domain; it reduces TopGen's test-set error by 10% 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 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), among many others, have shown that combining the output of several neural networks improves generalization over a single network. As an initial test, we combined the predictions of all the networks belonging to the final population by tak-

[4] 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.

[5] Lamarckian evolution is a theory based on the inheritance of characteristics acquired during a lifetime.

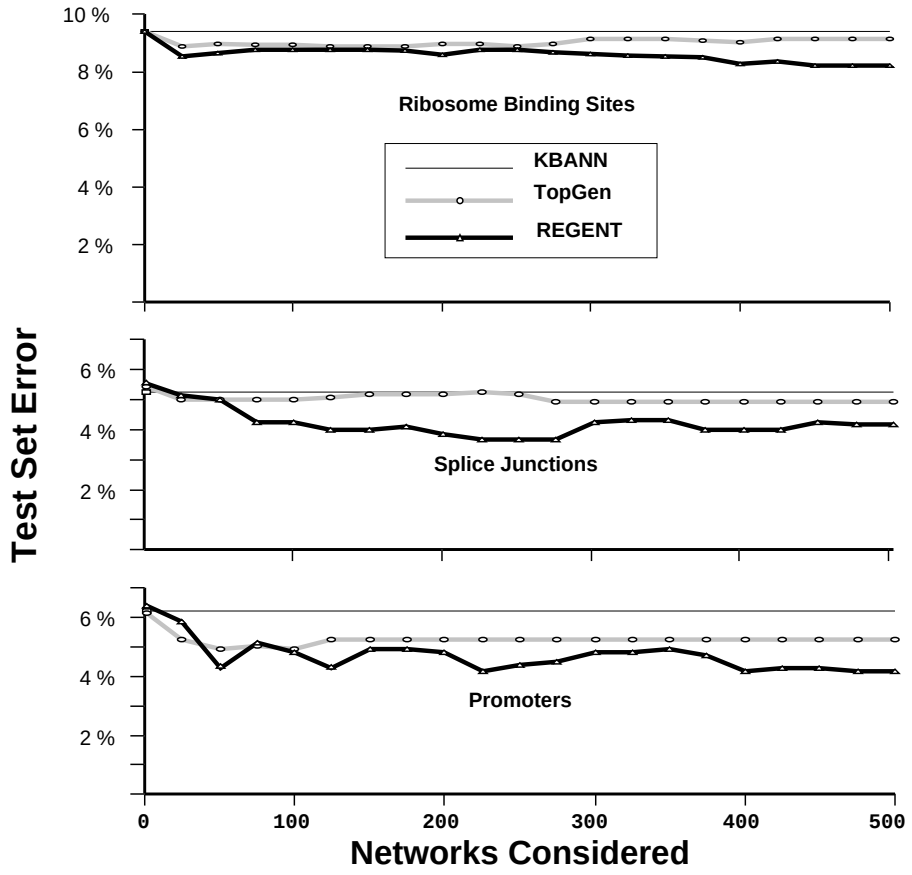


Figure 4: Error rates on three Human Genome problems.

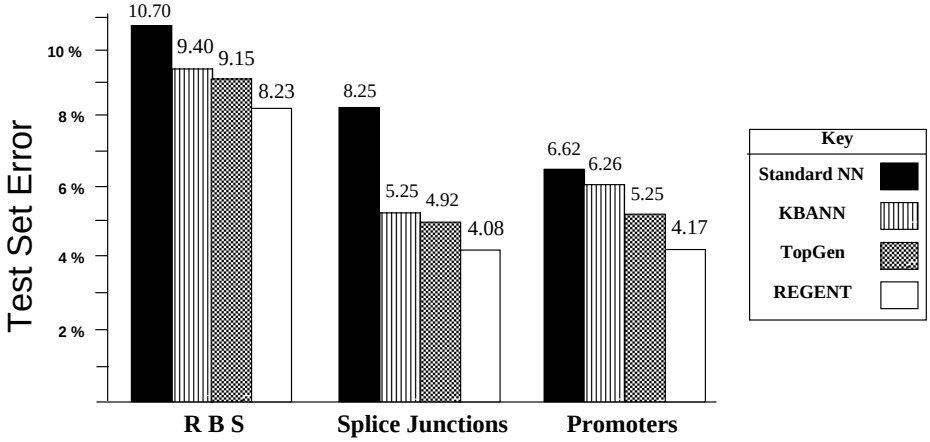


Figure 5: Error rates after TopGen and REGENT consider 500 networks. Two-sample one-tailed $t$-tests indicate that REGENT differs from both KBANN and TopGen at the 90.0% confidence level on all three domains.

ing the weighted-average of each network (as determined by its fitness). Simply combining the networks in this fashion produced test-set errors of 7.91% on the RBS domain, 3.42% on the splice-junction domain, and 3.49% on the promoter domain.

When combining multiple networks, Hansen and Salamon (1990) showed that an increase in generalization is likely if the individual networks tend to be independent in their errors. To help promote this independence, we plan to investigate incorporating diversity-promoting techniques (Goldberg, 1989), thus minimizing the similarity of the networks in our final population. Instead of replacing the network with the lowest validation-set error, we will replace the network that has both a poor validation-set performance *and* classifies examples similarly to other networks in the population. We plan to estimate the classification similarity between two networks with the examples in the validation set; hence, networks that classify the validation set in a similar fashion will have a low chance of survival. Another diversity-promoting alternative we plan to investigate is to create subpopulations (Deb & Goldberg, 1989), and then combine a network from each subpopulation.

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. In fact, this may be a possible explanation for the occasional upward trends in test-set error, from both TopGen and REGENT, in Figure 4. Future work, then, is to investigate selection methods that do not use a validation set, which would also allow us to use all the training instances to train the networks. Such techniques include minimum description length methods (Rissanen, 1983), Generalized Prediction Error (Moody, 1991), and Bayesian methods (MacKay, 1992).

Since the correct theory may be quite different 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 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 suitable for REGENT's crossover and mutation operators.

Finally, our future plans include using a rule-extraction algorithm to test the interpretability of a REGENT-refined network. Obtaining human-understandable rules would allow an expert to understand what has been learned. REGENT adds nodes, during its mutation, in a fashion that does not violate the two assumptions made in Towell and Shavlik's rule-extraction algorithm (1993), and during its crossover, REGENT tries to retain the rule structure of the network; therefore, we hypothesize that the networks generated by REGENT should be interpretable.

## 6 RELATED WORK

The most obvious related work is the KBANN and TopGen algorithms, which we described earlier.[6] Fletcher and Obradovic (1993) present an approach that also adds nodes to a KBANN network. Their approach constructs a single-layer of nodes, fully connected between the input and output units, "off to the side" of KBANN. Their approach does not change the weights of the KBANN portion of the network, so modifications to the initial rule base are left to the constructed hidden nodes. Also, this approach adds nodes until training set error is sufficiently small, thus producing only one possible network. TopGen compared favorably to a similar technique that also added nodes off to the side of KBANN (Opitz & Shavlik, 1993).

Other related work includes applications of GAs to neural networks. GAs have been applied in two different ways: (1) to optimize the connection weights in a fixed topology and (2) to optimize the topology of the network. Techniques that use only GAs to optimize weights (Whitley & Hanson, 1989; Montana & Davis, 1989) have compared competitively with gradient-based training algorithms; however, one problem with GAs is their inefficiency in fine-tuned local search, thus the scalability of these methods are in question (Yao, 1993). Kitano (1990b) presents a method that combines GAs with backpropagation. He does this by using the GA to determine the starting weights for a network, which is then refined by backpropagation. REGENT differs from this method in that it uses a domain theory to help determine each network's starting weights and genetically searches, instead, for an appropriate network topology.

Most methods that use GAs to optimize a network topology use backpropagation to train each network's weights. Of these methods, many directly encode each connection in the network (Miller et al., 1989; Oliker et al., 1992; Schiffmann et al., 1992). These methods are relatively straightforward to implement, and are good at fine tuning small networks (Miller et al., 1989); however, they do not scale well since they require very large matrices to represent large networks (Yao, 1993). Other techniques (Harp et al., 1989;

[6] The relationship between connectionist theory-refinement systems and purely symbolic ones has been extensively covered (Towell, 1991; Baffes & Mooney, 1993); thus we do not discuss it here.

Kitano, 1990a; Dodd, 1990) only encode the most important features of the network. These indirect encoding schemes can evolve different sets of parameters along with the network's topology and have been shown to have good scalability (Yao, 1993). REGENT differs from both the direct and indirect methods in that it does not explicitly encode its networks. Some techniques (Koza & Rice, 1991; Oliker et al., 1992) evolve both the architecture and connection weights at the same time; however, the combination of the two levels of evolution greatly increases the search space.

REGENT differs mainly from both GA and non-GA (Fahlman & Lebiere, 1989; Mezard & Nadal, 1989; Frean, 1990) network-growing algorithms in that REGENT is designed for knowledge-based neural networks. Thus REGENT uses domain-specific knowledge and symbolic rule-refinement techniques to aid in determining the network's topology and initial weight setting. A second difference is that most of these other algorithms restructure their network based solely on training set error, while REGENT minimizes validation set error. Also, REGENT differs from the non-GA methods in that REGENT performs a different search than hill-climbing.

## 7 CONCLUSION

An ideal inductive-learning algorithm should be able to exploit the available resources of computing power and domain-specific knowledge to improve its ability to generalize. KBANN (Towell & Shavlik, in press) has been shown to be effective at translating a domain theory into a neural network; however, KBANN suffers in that it does not alter its topology. TopGen (Opitz & Shavlik, 1993) improved the KBANN algorithm by using available computer power to search for effective places to add nodes to the KBANN network; however, we show 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 present a new algorithm, REGENT, that uses a specialized genetic algorithm 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 so, 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. Therefore, one can view REGENT as an anytime learner, which makes effective use of problem-specific knowledge and available computing cycles.

**Acknowledgements**

This work was supported by Department of Energy grant DE-FG02-91ER61129, Office of Naval Research grant N00014-93-1-0998, and National Science Foundation grant IRI-9002413. We would also like to thank Michiel Noordewier of Rutgers for creating the domain theories and data sets we used in this paper.

## References

Ackley, D. (1987). *A Connectionist Machine for Genetic Hillclimbing*. Kluwer, Norwell, MA.

Ackley, D. & Littman, M. (1994). A case for lamarckian evolution. In Langton, C., editor, *Artificial Life III*, (pp. 3–10), Redwood City, CA. Addison-Wesley.

Baffes, P. & Mooney, R. (1993). Symbolic revision of theories with M-of-N rules. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence*, (pp. 1135–1140), Chambery, France. Morgan Kaufmann.

Dean, T. & Boddy, M. (1988). An analysis of time-dependent planning. In *Proceedings of the Seventh National Conference on Artificial Intelligence*, (pp. 49–54), St. Paul, MN. Morgan Kaufmann.

Deb, K. & Goldberg, D. (1989). An investigation of niche and species formation in genetic function optimization. In *Proceedings of the Third International Conference on Genetic Algorithms*, (pp. 42–50), Arlington, VA. Morgan Kaufmann.

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.

Fahlman, S. & Lebiere, C. (1989). The cascade-correlation learning architecture. In Touretzky, D., editor, *Advances in Neural Information Processing Systems (volume 2)*. Morgan Kaufmann, San Mateo, CA.

Farmer, J. D. & Belin, A. (1992). Artificial life: The coming evolution. In Langton, C., Taylor, C., Farmer, J. D., & Rasmussen, S., editors, *Artificial Life II*, (pp. 815–840), Redwood City, CA. Addison-Wesley.

Fletcher, J. & Obradovic, Z. (1993). Combining prior symbolic knowledge and constructive neural network learning. *Connection Science*, 5:365–375.

Frean, M. (1990). The upstart algorithm: A method for constructing and training feedforward neural networks. *Neural Computation*, 2:198–209.

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

Ginsberg, A. (1990). Theory reduction, theory revision, and retranslation. In *Proceedings of the Eighth*

*National Conference on Artificial Intelligence*, (pp. 777–782), Boston, MA. AAAI/MIT Press.

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

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

Harp, S., Samad, T., & Guha, A. (1989). 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. (1975). *Adaptation in Natural and Artificial Systems.* University of Michigan Press, Ann Arbor, MI.

Kitano, H. (1990a). Designing neural networks using genetic algorithms with graph generation system. *Complex Systems*, 4:461–476.

Kitano, H. (1990b). Empirical studies on the speed of convergence of neural network training using genetic algorithms. In *Proceedings of the Eighth National Conference on Artificial Intelligence*, (pp. 789–795), Boston, MA. AAAI/MIT Press.

Koza, J. (1992). *Genetic Programming.* MIT Press, Cambridge, MA.

Koza, J. & Rice, J. (1991). Genetic generation of both the weights and architectures for a neural network. In *International Joint Conference on Neural Networks (volume 2)*, (pp. 397–404), Seattle, WA.

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

Litzkow, M., Livny, M., & Mutka, M. (1988). Condor — a hunter of idle workstations. In *Proceedings of the Eighth International Conference on Distributed Computing Systems.* Computer Society Press.

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

Mezard, M. & Nadal, J.-P. (1989). Learning in feedforward layered networks: The tiling algorithm. *Journal of Physics A*, 22:2191–2204.

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

Montana, D. & Davis, L. (1989). Training feedforward networks using genetic algorithms. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, (pp. 762–767), Detroit, MI. Morgan Kaufmann.

Moody, J. (1991). The *effective* number of parameters: An analysis of generalization and regularization in nonlinear learning systems. In Moody, J., Hanson, S., & Lippmann, R., editors, *Advances in Neural Information Processing Systems (volume 4)*, (pp. 847–854), San Mateo, CA. Morgan Kaufmann.

Oliker, S., Furst, M., & Maimon, O. (1992). A distributed genetic algorithm for neural network design and training. *Complex Systems*, 6:459–477.

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

Ourston, D. & Mooney, R. (1994). Theory refinement combining analytical and empirical methods. *Artificial Intelligence*, 66(2):273–309.

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

Rissanen, J. (1983). A universal prior for integers and estimation by minimum description length. *Annals of Statistics*, 11(2):416–431.

Schiffmann, W., Joost, M., & Werner, R. (1992). Synthesis and performance analysis of multilayer neural network architectures. Technical Report 16, University of Koblenz, Institute for Physics.

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

Towell, G. & Shavlik, J. (1993). Extracting refined rules from knowledge-based neural networks. *Machine Learning*, 13(1):71–101.

Towell, G. & Shavlik, J. (in press). Knowledge-based artificial neural networks. *Artificial Intelligence.*

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.

Whitley, D. & Hanson, T. (1989). Optimizing neural networks using faster, more accurate genetic search. In *Proceedings of the Third International Conference on Genetic Algorithms*, (pp. 391–396), Arlington, VA. Morgan Kaufmann.

Yao, X. (1993). Evolutionary artificial neural networks. *International Journal of Neural Systems*, 4(3):203–221.