Evolutionary Neural Networks for Value Ordering in Constraint Satisfaction Problems

David E. Moriarty and Risto Miikkulainen
Department of Computer Sciences
The University of Texas at Austin, Austin, TX 78712
moriarty,risto@cs.utexas.edu

Technical Report AI94-218
May 1994

Abstract

A new method for developing good value-ordering strategies in constraint satisfaction search is presented. Using an evolutionary technique called SANE, in which individual neurons evolve to cooperate and form a neural network, problem-specific knowledge can be discovered that results in better value-ordering decisions than those based on problem-general heuristics. A neural network was evolved in a chronological backtrack search to decide the ordering of cars in a resource-limited assembly line. The network required 1/30 of the backtracks of random ordering and 1/3 of the backtracks of the maximization of future options heuristic. The SANE approach should extend well to other domains where heuristic information is either difficult to discover or problem-specific.

1 Introduction

Constraint satisfaction problems (CSP) are common in many areas of computer science such as machine vision, scheduling, and planning. A CSP generally consists of a set of variables and a set of possible values for them. The variables must be bound such that none of the constraints in the problem are violated. For a survey of current CSP research see (Kumar 1992).

Most CSP methods are based on depth-first search with backtracking. When variables are instantiated, constraints are propagated forward, which either constrains the possible values for other variables or produces a contradiction. If a contradiction is found, the search backtracks and alternative variable bindings are tried. Clearly, choosing variable and value binding wisely can have a significant impact on the time required to find a solution.

Most CSP applications use the search-rearrangement or first-fail method (Bitner and Reingold 1975; Haralick and Elliot 1980) for ordering the variable bindings. At each level of the search, the variable with the smallest domain is chosen for instantiation. Purdom (1983) showed the search-rearrangement heuristic to be effective in a variety of CSP problems. Value-ordering heuristics, however, are more difficult to design and evaluate. Dechter and Pearl (1988) proposed a method in which the values that leave the CSP with the easiest solution are preferred. Their method involves estimating the difficulty of a CSP by translating the constraint graph into a tree-structure and counting the solutions. Another value-ordering method, the maximization of future options

heuristic, prefers values that leave the largest number of options open for future variable assgnments (Haralick and Elliot 1980; Kumar 1992). Using a similar heuristic, Kale (1990) was able to solve an order of magnitude larger instances of the n-queens problem than using the standard left-right column ordering.

One problem with devising good value-ordering heuristics is that they can be highly problem specific (Kumar 1992). For example, Sadeh (1991) showed that Dechter and Pearl's approach, which is problem general, performs poorly on job-shop scheduling problems where problem-specific heuristics generally do well. It would thus be significant if such specific heuristics could be developed automatically for each problem.

In the experiments reported in this paper, a new method called SANE (Symbiotic, Adaptive Neuro-Evolution) was used to discover neural networks that make good value-ordering decisions in a given problem. SANE forms neural networks by evolving, through genetic algorithms, individual neurons that exhibit symbiotic behavior. Artificial neural networks have proven very effective in pattern recognition and pattern association tasks, which makes them a good candidate for recognizing situations where value-ordering decisions can greatly affect solution time. Genetic algorithms provide a powerful, general training tool for neural networks in which no previous knowledge of the task is needed. Since the neural network extracts its knowledge from direct interactions with the CSP, it can discover problem-specific, heuristic information that can lead to more efficient constraint satisfaction search.

Using SANE, a neural network was evolved to decide the ordering of classes of cars on an assembly line, which is an NP-Complete problem (Van Hentenryck et al. 1992). After evolution, the number of backtracks incurred before a solution was found was compared with random value ordering and the maximization of future options heuristic. The network required 1/30 of the backtracks of random value ordering and 1/3 of the backtracks of the maximization of future options heuristic.

The car sequencing problem is briefly described below in Section 2. Section 3 presents the implementation of evolutionary neural networks for value ordering in a chronological backtrack search. Section 4 and 5 describe the SANE method for evolving neural networks and present the specifics of the evolution simulations. Comparisons of the SANE network with random value ordering and the maximization of future options heuristic are presented in section 6. As a conclusion, future directions for genetic decision networks are discussed in section 7.

2 The Car Sequencing Problem

Car sequencing is an instance of the general job-shop scheduling problem (Fox 1987). In an automobile factory, a continuously moving assembly line is used to put options such as power windows and sun roofs on cars. When a car enters an option station, the workers walk along with the car until the option has been installed. Some options take longer to install than others. The capacity of the option station is indicated by "r out of s": For example, an option station that has a capacity of 2 out of 5 can handle a maximum of 2 cars for every 5 that pass on the assembly line. If 3 cars require that option, the option station is said to be overdriven.

Different classes of cars require different options. The problem is to find an ordering of cars on the assembly line such that no option station is overdriven. In a CSP formulation, the slots represent the variables and the car classes represent the possible values for the variables. Previous approaches to this problem have used constraint logic programming to perform a depth-first search

Classes	1	2	3	4	5	6	Capacity (r/s)
Option 1	+	-	-	-	+	+	1/2
Option 2	-	-	+	+	-	+	2/3
Option 3	+	-	-	-	+	-	1/3
Option 4	+	+	-	+	-	-	2/5
Option 5	-	-	+	-	-	-	1/5

Table 1: The car-sequencing problem with 6 classes and 5 options. The options required by each class are indicated with a +. The capacities are shown in the form of r cars out of s slots (r/s).

Slots	1	2	3	4	5	6	7	8
Class 1	+	-	-					
Class 2	-							
Class 3	-							
Class 4	-							
Class 5	-	-	-					
Class 6	-	-						

Table 2: A partial solution to a problem with 8 cars. The assignment of class 1 (chosen arbitrarily) to slot 1 is indicated by the +. The constraints after instantiation of the first slot are shown by -. Classes 1 and 5 cannot be placed in slots 2 or 3 because of the capacity limit of option station 3. Since class 1 (assigned to slot 1) requires option 3 and option station 3 has a capacity of 1 out of 3 slots, no class that requires option 3 may be placed in slots 2 or 3. Class 6 cannot be placed in slot 2 because of the capacity of option 1. The slot with most constrained domain is slot 2: it can only be instantiated with classes 2, 3, or 4.

(Van Hentenryck et al. 1992) and automated reasoning to produce near-optimal solutions (Parrello et al. 1986).

Table 1 shows a particular car sequencing problem taken from (Van Hentenryck et al. 1992). The number of classes, number of options, capacities of the option stations, and options required by each class were fixed. The number of each cars in each class and total number of cars to schedule were varied in different instances of the problem.

In our implementation, the first-fail heuristic was used to decide the order of slot instantiations. Thus, at each level of the search, the slot with the smallest domain is chosen for instantiation. Slot 1 is designated (arbitrarily) as the first slot to instantiate because initially there are no constraints and all domains are equally large. Once slot 1 is instantiated, the slot with the smallest domain will be slot 2, since any capacity constraints on later slots must also be in effect for slot 2 (table 2). Similarly, after instantiating slot 2 and subsequent slots, the next slot on the assembly line will always have the smallest domain. In other words, the first-fail heuristic results in instantiating the slots in increasing order. However, it is not as easy to decide the order in which the different car classes should be assigned to the slots.

3 Genetic Decision Networks for Value Ordering

In a sequential decision task (Barto et al. 1990; Grefenstette et al. 1990), an agent observes a state of the system and chooses from a finite set of actions. The system then enters a new state upon

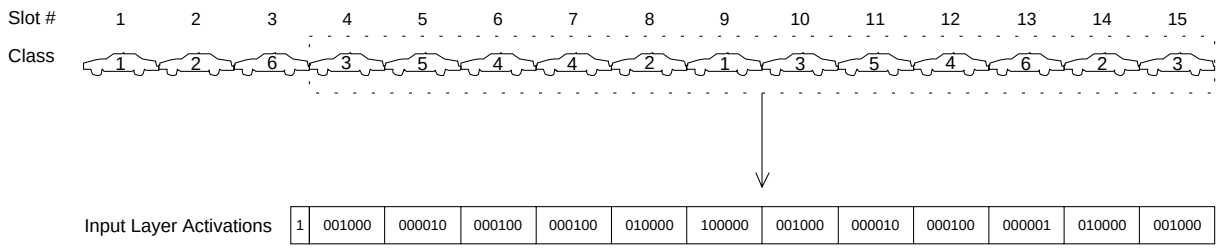

Figure 1: A partial sequence instantiation and the corresponding input to the network. The network receives the previous 12 assignments as input. For example, a car of class 3 has been assigned to slot 4. The first input unit is always 1 to allow the network to produce good initial choices. The next slot to be scheduled is slot 16.

which the agent must select another action. The system may return a payoff for each decision made or a payoff for a sequence of decisions. The objective is to choose the sequence of decisions such that the total payoff is maximized.

Previously, evolutionary neural networks have been shown to be very good at sequential decision tasks such as selecting game moves and focusing minimax search (Moriarty and Miikkulainen 1993, 1994a). The usual neural network learning algorithms such as backpropagation (Rumelhart et al. 1986) are impractical in such tasks since they require that the correct decision is known at each step. Such information is very difficult to establish in sequential decision tasks, because it is not always obvious how individual decisions affect the final outcome. In the neuro-evolution approach, however, evolutionary pressure guides the networks toward wise decisions. Networks that make better decisions receive higher fitness scores, allowing their genes to survive and propagate to future generations.

In the car sequencing problem, a genetic decision network was evolved to decide which type of car to place in the next slot on the assembly line. The network was implemented as part of a chronological backtrack search program. At each level of the search, the network received a window of the 12 previous slot assignments as input (figure 1). Each slot was represented by six input units, one for each car class (i.e. for each value assignment for the slot). Initially, all the input units would be 0 because no assignments have been made. Since the neural network needs some activation in the input layer to produce output, an extra (bias) input unit which was always 1 was included to allow the network to generate initial choices. The entire input layer, thus, consisted of 73 units. Figure 1 shows an example instantiation of the assembly line and the input the network receives.

The output layer consisted of six units, one for each class. The activation of each output unit (computed as a weighted sum of its input activations) indicates how strongly the network suggests assigning that class to the next slot. The output layer, thus, represents a ranking of the classes and determines the order in which classes are assigned to the slots during search, unless the choice violates either of the following two constraints:

1. There must be a car of that class remaining to be assigned, and
2. The assignmentmust must not violate any option station's capacity.

The network has no knowledge of the number and types of cars to schedule. Its output layer merely represents the order in which values should be tried given the current slot assignments. If there are no cars left of the highest ranked class or the assignment would cause an option station

to be overdriven, the class with the next highest output unit is tried, unless it too violates one of the two constraints.

Implementing these two simple constraints outside the network serves to *essentialize* the problem and relieve it of much of the trivial overhead. The primary task is to differentiate between good and bad choices. By not requiring the network to identify which classes are valid, it can more easily learn the value-ordering task. This approach is analogous to removing the requirement of legal move identification from a move-evaluating network in game playing, which also proved to be a good strategy (Moriarty and Miikkulainen 1993).

A simple forward-checking algorithm was also implemented to prune the search space early. For each option station, the total number of cars requiring that option was counted. If the number exceeds the capacity of the option station over all remaining slots, the search path was terminated.

4 Symbiotic Evolution with SANE

The value-ordering network was evolved using a novel neuro-evolution method called (SANE; Moriarty and Miikkulainen 1994b). Most neuro-evolution methods operate on a population of neural networks, where the fitness of each network is determined independently of other networks in the population. Unfortunately, this approach can often cause the population to prematurely converge to a single dominant network. Instead of multiple parallel searches through the encoding space, the search becomes a random walk using the mutation operator. Several approaches have been developed to help keep diversity in the population. These include fitness sharing (Goldberg 1989), adaptive mutation (Whitley et al. 1990), crowding (Dejong 1975), and local mating (Collins and Jefferson 1991; Davidor 1991). Each of these techniques relies on external genetic functions that prevent convergence of the genetic material.

SANE achieves population diversity by making it an essential part of the task. SANE evolves a *population of neurons*, where each neuron's task involves making connections with other neurons in the population to form a functioning neural network. The fitness of each neuron is determined by how well it cooperates with other neurons in the population. To evolve a network capable of performing a task, the neurons must form a symbiotic relationship. Since no one neuron will perform well alone, evolution will guide the population towards diverse, symbiotic neurons. Premature convergence is thus avoided, and the population can discover better solutions to more difficult problems.

The basic steps in one generation of SANE are as follows (table 3): During the evaluation stage, random subpopulations are selected and combined to form a neural network. The network is evaluated in the task and assigned a score. The score is added to each selected neuron's fitness value. The process continues until each neuron has participated in a sufficient number of networks. The average fitness value of each neuron is then computed by dividing the sum of its fitness values by the number of networks it participated in. The neurons that have a high average fitness value have cooperated well with other neurons in the population. Networks that do not cooperate and are detrimental to the networks that they participate in receive low fitness evaluations and are selected against. Once each neuron has an average fitness value, crossover operations are used to combine the chromosomes of the neurons with the best average fitness values. Mutation is also used to prevent the loss of key genetic material.

In the current implementation, each neuron is defined in a bitwise chromosome that encodes a series of connection definitions. Each definition consisted of an 8-bit label field and a 16-bit weight

1. Clear all neuron fitness values.
2. Select ζ neurons randomly from the population.
3. Create a neural network from the selected neurons.
4. Evaluate the network in the given task.
5. Add the network's score to each selected neuron's fitness value.
6. Repeat steps 2-5 a sufficient number of times.
7. Calculate each neuron's average fitness value by dividing its total fitness value by the number of networks it participated in.
8. Perform genetic operations on the population based on the fitness value of each neuron.

Table 3: The basic steps in SANE.

field. The absolute value of the label determines where the connection is to be made. The neurons connect only to the input and the output layer. If the decimal value of the label, D, is greater than 127, then the connection is made to output unit D mod O, where O is the total number of output units. Similarly, if D is less than or equal to 127, the connection is made to input unit D mod I, where I is the total number of input units. The weight field encodes a floating point weight value for the connection.

5 Evolution Simulations

A population of 800 linear threshold neurons, each with a threshold of 0, was evolved to decide value ordering in the car sequencing problem. The subpopulation size ζ was 100, and 40 networks were formed during each generation of neurons. Each neuron thus participated in an average of 5 networks per generation. The neurons were encoded in 240-bit chromosomes which contained ten 24-bit connection definitions.

To evaluate each network, 5 scheduling problems were selected from a database of 1000 problem instances and the network was used to perfrom class ordering in a chronological backtrack search. The problem instances contained between 10 to 25 cars. The option requirements and option station capacities were as shown in table 1. The number of backtracks during the search was used as the score for each network.

After the evaluation the neurons were sorted in non-decreasing order of their fitness values. The top 200 neurons were genetically combined using a two-point crossover operation, giving each offspring the beginning and tail of one parent's chromosome and the middle of the other parent's. A mate for each of the top neurons was selected randomly among all neurons with equal or higher average fitness values. Two offspring were formed per mating, resulting in 400 new neurons per generation. The new neurons replaced the 400 worst neurons in the population. Mutation at the rate of 0.1% was performed on the entire population as the last step in each generation.

6 Results

The population was evolved for 100 generations requiring approximately 40 minutes on an IBM RS6000 25T. The best network in each generation was evaluated using a 50 problem validation set.

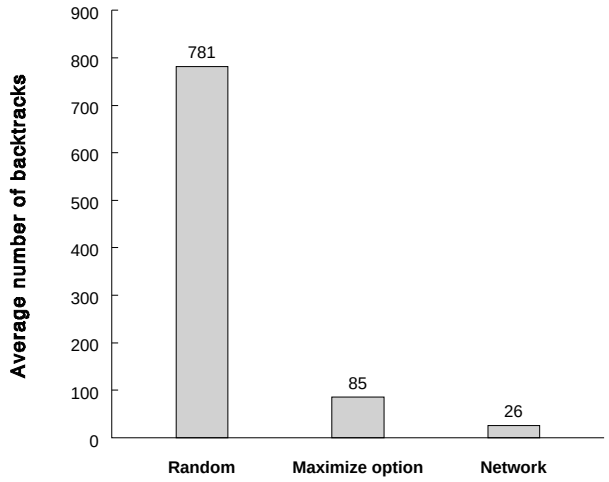

Figure 2: The average number of backtracks for each problem in the 50 problem test set.

As the final result, the best network over all generations was selected and tested on a different 50 problem test set. For comparison, random value ordering and the maximization of future options heuristic (Haralick and Elliot 1980; Kale 1990) were also run on the test set. The maximization of future options heuristic was implemented to prefer the class that leaves the most option stations free.

Figure 2 shows the average number of backtracks for each problem for random value ordering, the maximization of future options heuristic, and the SANE network. The graph clearly shows the advantage of problem-specific knowledge acquired through the SANE method. While the problem-general heuristic did reduce the number of backtracks significantly over random ordering, it required 3 times more backtracks than the SANE network.

Figure 3 illustrates the actual choices the network was making. The graph shows the distribution of the classes for each rank; for example, class 1 was ranked first 100% of the time in the network's output and class 4 second 91% of the time. As seen from the graph, the network suggested the class ordering 1,4,6,5,3,2 the majority of the time, with positions 3 and 4 having the most variability. Thus, the network appears to be taking a first-fail approach to value ordering by preferring classes that place the most demand on the system through their option requirements. This approach is most obvious in the case of class 1, and constitutes the biggest difference between the network's ordering and that of the maximization of future options heuristic. The network always preferred to schedule cars of class 1 as soon as possible, whereas the maximization of future options heuristic normally tried it last. Intuitively, cars of class 1 should be difficult to schedule, because they require the most options. Thus, it seems sensible that if a car of class 1 needs to be scheduled and it can fit without causing any immediate conflicts, it should be placed in the next slot. The maximization of future options heuristic, however, will not schedule it because it will limit the remaining options available to future cars. This approach delays the scheduling of class 1 cars and can incur large backtracks if they cannot fit later. The maximization of future options is normally a good approach because it directs the search toward areas in the search space with high solution densities (Kale 1990). In this particular case, however, SANE discovered a better ordering through problem-specific knowledge.

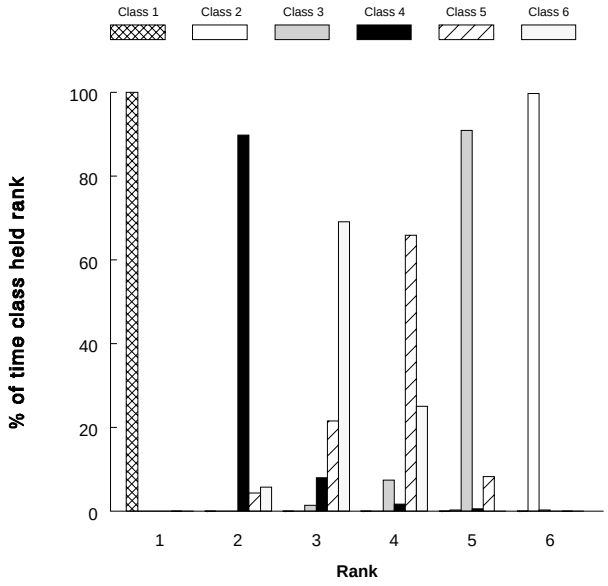

Figure 3: The distribution of the network's choices for each rank.

7 Discussion and Future Work

The search strategy that the value-ordering network was embedded in was a simple, chronological, depth-first search. Chronological search, however, can be inefficient because it can lead to numerous recombinations of variable bindings that together cause failure. This is commonly known as thrashing (Mackworth 1977). A more intelligent approach is to use dependecy-directed backtracking (Stallman and Sussman 1977; Dechter 1990), where the search backtracks to the variable binding that caused failure. Despite the improvements to the backtracking algorithm, the search remains vulnerable to poor value ordering. Genetic decision networks should therefore similarly increase efficiency in more intelligent backtracking schemes.

The constraint propagation algorithm could also be improved. For example, Van Hentenryck et al. (1992) implemented additional "surrogate" constraints in the car sequencing problem to prune large areas of the search space. Surrogate constraints are redundant, that is, they do not further constrain the problem. Operationally, however, they can make the search space much smaller by exploiting properties that must be satisfied by all solutions. Since the decision networks adapt directly to the search environment, the networks would learn to make use of the surrogate constraints as well in their decision evaluation.

Using a sliding window of the previous 12 slots for the input assumes that effective decisions can be based on a limited view of the problem. The 12-slot window appeared to perform well in this problem, but there is no guarantee that for larger problems, good decisions could still be made. If an upper bound on the input size is known, at least approximately, an appropriate window size can be chosen. Another possible solution is to encode the input space with continuous values. A simple recurrent network could be used to superimpose representations on the same set of real-valued units. Similar sequences would have similar representations, which should help the network generalize to new and larger sequences.

Each of the problem instances was feasible in that no option station needed to be overdriven. In a real assembly plant, however, it may be necessary to schedule a set of cars even when no feasible

schedules exist. In this case, which stations to overdrive becomes the central decision. Parrello et al. (1986) proposed a method based on overdrive penalties. Stations that could recover quickly produced lower penalties than stations that became saturated. The problem thus reduces to finding a schedule with the minimum total penalty, which is an NP-hard problem. Evolutionary neural networks could be very effective in such minimization problems as well. Instead of the number of backtracks, the total overdrive penalty would determine the fitness of the networks. The genetic search minimizes the fitness function, and networks that produce low average overdrive penalties would evolve.

Genetic decision networks perform best when their output units represent a ranking of the possible choices based only on the current activation of the network. The networks are less successful when their decisions must be based on absolute values or previous activations. For example, Moriarty and Miikkulainen (1994a) showed that the decision networks were very good at choosing the best path for a minimax search. Such decisions can be effectively made based only on the current state and the current network activation. Evolutionary neural netorks, however, performed poorly when evolved to evaluate board positions and to produce an absolute value of goodness for each board. This task requires decisions that are based on global information, namely the evaluation of other boards in other parts of the search tree. Thus, to make effective evaluations the network would have to take into account previous activations (board evaluations) before assigning an absolute value. Such behavior has proven very difficult to evolve.

We are currently studying the application of genetic decision networks to a broad range of domains including very complex systems such as communication networks and neurocontrol problems. Genetic decision networks should be able to optimize virtually any decision point through direct interactions with the system or a model of the system. Some tasks where decision networks could prove useful include routing, scheduling, parsing, and control of discrete systems.

8 Conclusion

Neural networks, evolved through the SANE method, provide a new mechanism for value-ordering in constraint satisfaction problems. Problem-specific, heuristic knowledge is discovered through direct interaction with the problem and implemented in a neural network that can generalize to new situations. In the car sequencing problem, a decision network was evolved that reduced the number of backtracks significantly over a problem-general heuristic. Genetic decision networks should similarly increase efficiency in a variety of other domains where good decision strategies are not easily realized.

Acknowledgements

Thanks to Othar Hansson for suggesting constraint satisfaction as a good domain for genetic decision networks.

References

Barto, A. G., Sutton, R. S., and Watkins, C. J. C. H. (1990). Learning and sequential decision making. In Gabriel, M., and Moore, J. W., editors, *Learning and Computational Neuroscience.*

Cambridge, MA: MIT Press.

Bitner, J., and Reingold, E. M. (1975). Backtrack programming techniques. *Communications of the ACM*, 18:651–655.

Collins, R. J., and Jefferson, D. R. (1991). Selection in massively parallel genetic algorithms. In *Proceedings of the Fourth International Conference on Genetic Algorithms*, 249–256. San Mateo, CA: Morgan Kaufmann.

Davidor, Y. (1991). A naturally occurring niche and species phenonenon: the model and first results. In *Proceedings of the Fourth International Conference on Genetic Algorithms*, 257–263. San Mateo, CA: Morgan Kaufmann.

Dechter, R. (1990). Enhancement schemes for constraint posting: Backjumping, learning, and cutset decomposition. *Artificial Intelligence*, 41(3):273–312.

Dechter, R., and Pearl, J. (1988). Network-based heuristics for constraint satisfaction problems. *Artificial Intelligence*, 34:1–38.

Dejong, K. A. (1975). *An Analysis of the Behavior of a Class of Genetic Adaptive Systems.* PhD thesis, The University of Michigan, Ann Arbor, MI.

Fox, M. S. (1987). *Constraint-Directed Search: A Case Study of Job-Shop Scheduling.* San Mateo, CA: Morgan Kaufmann.

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

Grefenstette, J. J., Ramsey, C. L., and Schultz, A. C. (1990). Learning sequential decision rules using simulation models and competition. *Machine Learning*, 5:355–381.

Haralick, R., and Elliot, G. (1980). Increasing tree search efficiency for constraint satisfaction problems. *Artificial Intelligence*, 14(3):263–313.

Kale, L. V. (1990). A perfect heuristic for the n non-attacking queens problem. *Information Processing Letters*, 34(4):173–178.

Kumar, V. (1992). Algorithms for constraint satisfaction problems: A survey. *AI Magazine*, 13:32–44.

Mackworth, A. K. (1977). Consistency in networks of relations. *Artificial Intelligence*, 8:99–118.

Moriarty, D. E., and Miikkulainen, R. (1993). Evolving complex Othello strategies using marker-based genetic encoding of neural networks. Technical Report AI93-206, Department of Computer Sciences, The University of Texas at Austin.

Moriarty, D. E., and Miikkulainen, R. (1994a). Evolving neural networks to focus minimax search. In *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94).* Seattle, WA.

Moriarty, D. E., and Miikkulainen, R. (1994b). Neural network reinforcement learning through symbiotic evolution. Technical Report AI94-224, Deptartment of Computer Sciences, The University of Texas at Austin.

Parrello, B. D., Kabat, W. C., and Wos, L. (1986). Job-shop scheduling using automated reasoning: A case study of the car-sequencing problem. *Journal of Automated Reasoning*, 2:1–42.

Purdom, P. (1983). Search rearrangement backtracking and polynomial average time. *Artificial Intelligence*, 21:117–133.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning internal representations by error propagation. In Rumelhart, D. E., and McClelland, J. L., editors, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Volume 1: Foundations*, 318–362. Cambridge, MA: MIT Press.

Sadeh, N. (1991). Look-ahead techniques for micro-opportunistic job shop scheduling. ph.d. diss. Technical Report CMU-CS-91-102, Computer Science Department, Carnegie Mellon University.

Stallman, R., and Sussman, G. J. (1977). Forward reasoning and dependency-directed backtracking. *Artificial Intelligence*, 9(2):135–196.

Van Hentenryck, P., Simonis, H., and Dincbas, M. (1992). Constraint satisfaction using constraint logic programming. *Artificial Intelligence*, 58:113.

Whitley, D., Starkweather, T., and Bogart, C. (1990). Genetic algorithms and neural networks: Optimizing connections and connectivity. *Parallel Computing*, 14:347–361.