In V. Honavar, M. Patel, and K. Balakrishnan (editors)
Advances in the Evolutionary Synthesis of Neural Systems
Cambridge, MA: MIT Press, 1999.

Learning Sequential Decision Tasks

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 AI95-229
January 1995

Abstract

This paper presents a new approach called SANE for learning and performing sequential decision tasks. Compared to problem-general heuristics, SANE forms more effective decision strategies because it learns to utilize domain-specific information. SANE evolves neural networks through genetic algorithms and can learn in a wide range of domains with minimal reinforcement. SANE's evolution algorithm, called symbiotic evolution, is more powerful than standard genetic algorithms because diversity pressures are inherent in the evolution. SANE is shown to be effective in two sequential decision tasks. As a value-ordering method in constraint satisfaction search, SANE required only 1/3 of the backtracks of a problem-general heuristic. As a filter for minimax search, SANE formed a network capable of focusing the search away from misinformation, creating stronger play.

1 Introduction

Sequential decision tasks (Barto et al. 1990; Grefenstette et al. 1990) can be characterized by the following scenario: An agent observes a state of a dynamic system and chooses from a finite set of actions. The system then enters a new state upon which the agent must select another action. The system may return a payoff after each decision made or after a sequence of decisions. The objective is to select the sequence of actions that return the highest cumulative payoff. Often the best strategy is not to maximize each individual payoff, because some actions may produce high immediate payoffs but may enter states from which high later payoffs are impossible. Developing good strategies involves assigning credit to individual decisions based on the overall payoff, which is a difficult problem in machine learning known as the *credit assignment problem.*

Sequential decision tasks appear in many practical real-world problems including control, game playing, scheduling, and resource allocation. In many current applications decisions are based on simple heuristics or "rule of thumb" strategies. These strategies are normally problem-general and do not take advantage of domain-specific information. For example, in a communication network, packet routing is normally decided by a shortest-path algorithm, which is a problem-general policy. However, Littman and Boyan (1993) have shown that better routing policies can be achieved by incorporating more domain-specific knowledge such as network topology and reliability of local information. Similarly, in most high-rise buildings, better elevator dispatching systems are being built by incorporating knowledge of the specific traffic patterns on each floor. Therefore, it appears that a learning mechanism to automatically develop such domain-specific strategies could be of great benefit in a wide range of tasks.

This paper describes a new approach called SANE (Symbiotic, Adaptive Neuro-Evolution) that combines genetic algorithms and neural networks to learn and perform sequential decision tasks. Neural networks have proven very effective in pattern recognition and pattern association tasks and have been shown to generalize well to unseen situations. Genetic algorithms provide a general training tool in which few assumptions about the domain are necessary. Since genetic algorithms only require a single fitness evaluation over the entire (possibly multi-step) task, they can be applied to domains with very sparse reinforcement, which makes them particularly well-suited for evaluating performance in sequential decision tasks.

SANE was implemented in two sequential decision tasks in the field of artificial intelligence. In the first task, SANE evolved a network to perform value ordering in a constraint satisfaction problem. The SANE network required 1/30 of the backtracks of random value ordering and 1/3 of the backtracks of the commonly-used maximization of future options heuristic. In the second task, SANE was implemented to focus a minimax search in the game of Othello. In this task, SANE formed a network to decide which moves from a given board situation are promising enough to evaluate. Using the powerful evaluation function from the Bill program (Lee and Mahajan 1990), SANE was able to generate better play while examining 33% fewer board positions than normal, full-width minimax search. SANE's performance in these domains demonstrates both its effectiveness and applicability to a broad range of tasks.

The body of this paper is organized as follows. In the next section, the basic steps in SANE are described along with details of the current implementation. Sections 3 and 4 present the empirical results in the tasks of value ordering and focusing a minimax search. Future research directions are outlined in section 5, which include applications to non-Markovian tasks and domains with multiple decision tasks.

2 Symbiotic, Adaptive Neuro-Evolution

Recently there has been much interest in combining genetic algorithms and neural networks. Genetic algorithms are global search techniques patterned after Darwin's theory of natural evolution. Numerous potential solutions are encoded in strings called *chromosomes* and evaluated in a task. Substrings, or *genes*, of the best solutions are combined to form new solutions, which are inserted into the population. Each iteration of the genetic algorithm consists of solution evaluation and recombination and is called a *generation*. The idea is that structures that led to good solutions in previous generations can be combined to form better solutions in subsequent generations.

In neuro-evolution, the solutions take the form of neural networks. Most approaches to neuro-evolution operate on a population of complete neural networks that are encoded in separate chromosomes (Belew et al. 1991; Koza and Rice 1991; Whitley et al. 1993). By evolving full solutions to the problem (i.e. complete neural networks), the algorithm typically converges the population towards a single dominant individual. Such concentration is desirable if it occurs at the global optimum, however, often populations *prematurely converge* to a local optimum. Once the population has converged,the search becomes a random walk using the mutation operator.

The problem of premature convergence is not unique to neuro-evolution, but is an open research issue in the genetic algorithms community as well. To prevent premature convergence, the population must remain diverse. Diversity will disperse individuals throughout the search space giving the population a more global view of the space of solutions. As a result, convergence at suboptimal solutions is much more unlikely.

SANE incorporates the idea of diversity into neuro-evolution. SANE evolves a population of neurons, where 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 optimize different aspects of the network and form a mutualistic symbiotic relationship. Neurons will evolve into several *specializations* that search different areas of the solution space. Premature convergence is thus avoided and the population can discover better solutions to more difficult problems.

SANE evolves a population of hidden neurons for a given type of architecture such as a 2-layer-feedforward network (2 layers of weights). The basic steps in one generation of SANE are as follows (table 1): During the evaluation stage, random subpopulations of size ζ are selected and combined to form a neural network. The network is evaluated in the task and assigned a score, which is subsequently added to each selected neuron's fitness variable. The process continues until each neuron has participated in a sufficient number of networks. The average fitness of each neuron is then computed by dividing the sum of its fitness scores by the number of networks in which it participated. The neurons with high average fitness have cooperated well with other neurons in the population. Neurons that do not cooperate and are detrimental to the networks that they form receive low fitness scores and are selected against.

Once each neuron has a fitness value, crossover operations are used to combine the chromosomes of the best-performing neurons. Mutation is employed at low levels to introduce new genetic material. In this sense, mutation is only used as an insurance policy against missing key genetic material, not as a mechanism to create diversity.

Each neuron is defined in a bitwise chromosome that encodes a series of connection definitions, each consisting of an 8-bit label field and a 16-bit weight field. The absolute

1. Clear all fitness values from each neuron.
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. Get the neurons' average fitness scores by dividing their total fitness values by the number of networks they were implemented in.
8. Perform crossover operations on the population based on the average fitness value of each neuron.

Table 1: The basic steps in one generation of SANE.

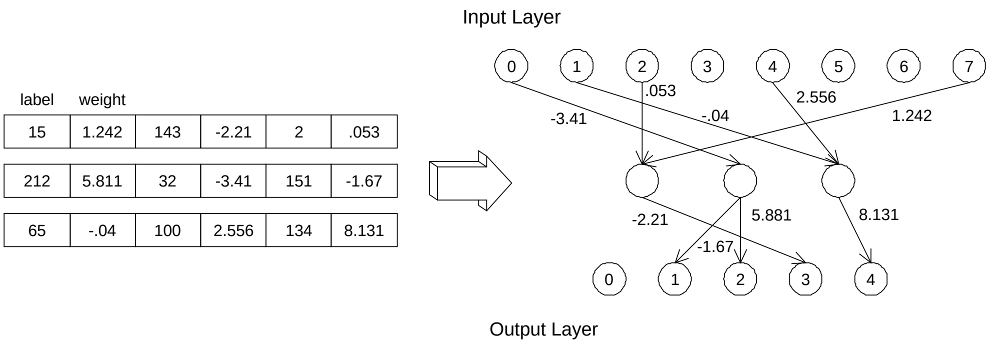

Figure 1: Forming a simple 8 input, 3 hidden, 5 output unit neural network from three hidden neuron definitions. The chromosomes of the hidden neurons are shown to the left and the corresponding network to the right. In this example, each hidden neuron has 3 connections.

value of the label determines where the connection is to be made. The neurons only connect 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, then 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 for the connection. Figure 1 shows how a neural network is formed from three sample hidden neuron definitions.

Once each neuron has participated in a sufficient number of networks, the population is ranked according to the average fitness values. A mate is selected for each neuron in the top quarter of the population by choosing a neuron with an equal or higher average fitness value. A one-point crossover operator is used to mate two neurons creating two offspring per mating. The two offspring replace the worst-performing neurons (according to the rank) in the population. Mutation at the rate of 1% is performed on the entire population as the last step in each generation.

The implementation of SANE outlined above has performed well, however, SANE could be implemented with a variety of different neuron encodings and even with network architectures that allow recurrency. More advanced encodings and evolutionary strategies may enhance both the search efficiency and generalization ability. Extensions to the current implementation will be a subject of future research.

An empirical evaluation of SANE was performed in the standard reinforcement learning benchmark of balancing a pole on a cart (Moriarty and Miikkulainen 1994a). The learning speed and generalization ability of SANE was compared to those of the best-known reinforcement learning approaches to this problem: the single-layer Adaptive Heuristic Critic (AHC) of Barto et al. (1983), the two-layer Adaptive Heuristic Critic of Anderson (1989) and the GENITOR neuro-evolution system of Whitley et al. (1993). SANE was found to be considerably faster (in CPU time) and more efficient (in training episodes) than the two-layer AHC and GENITOR implementations. Compared to the single-layer AHC, SANE was an order of magnitude faster even though it required more training episodes. The generalization capabilities of the four methods were comparable. An analysis of the final populations verifies that SANE finds solutions in diverse, unconverged populations and can maintain diversity in prolonged evolution.

The purpose of this paper is to show how SANE can be applied to existing and novel sequential decision tasks. The first task, value ordering in constraint satisfaction problems, was selected because it is an important problem where general solutions have performed inconsistently. The second task, focusing minimax search, was selected to demonstrate how SANE can be applied to existing problems previously unrealized as sequential decision tasks.

3 CSP Value Ordering

Constraint satisfaction problems (CSP) are common in many areas of computer science such as machine vision, scheduling, and planning. A number of variables must be assigned values such that none of the constraints among them are violated. CSPs are usually solved through depth-first search: the variables are chosen for instantiation one at a time, and their different legal values are tried one at a time. The order in which variables and values are considered determines how soon a solution is found, and therefore, choosing the variable and value bindings wisely can significantly reduce search time.

Most CSP applications use the first-fail method (Haralick and Elliot 1980) for ordering the variable bindings. At each level of the search, the variable with the smallest number of possible values is chosen for instantiation. However, deciding the order in which the values are assigned is much more difficult, partly because good value-ordering heuristics are highly problem specific (Kumar 1992). Learning the domain-specific heuristic information to build an effective value-ordering policy would therefore be a significant demonstration of SANE in an important sequential decision task.

3.1 The Car Sequencing Problem

Car sequencing is an instance of the job-shop scheduling problem (Van Hentenryck et al. 1992). In an automobile factory, a continuously moving assembly line is used to put options

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 2: The car-sequencing problem with 6 classes and 5 option stations. The options required by each class are indicated with a +. The capacities of the option stations are shown in the form r/s.

such as power windows on cars. When a car enters an option station, the workers walk along with the car until the option has been installed. The capacity of the option station is indicated by "r out of s": For example, an option station with 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 will be overdriven. Different classes of cars require different options. The problem is to find an ordering on the assembly line such that no option station becomes overdriven.

Table 2 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 cars in each class and total number of cars to schedule were varied in different instances of the problem. In a constraint satisfaction formulation, the slots on the assembly line represent the variables and the classes represent the possible values for the variables. In our experiments, the first-fail heuristic was used for variable ordering, which results in always assigning each slot in the order they appear on the assembly line (Moriarty and Miikkulainen 1994b). A good strategy for value ordering was left to be developed by SANE.

3.2 Evolving a Value-ordering Neural Network

A 2-layer neural network was evolved using SANE to decide which car class to place in the next slot on the assembly line. Networks were evaluated by implementing them as part of a chronological backtrack search program. At each level of the search, the network received a window of the previous 12 slot assignments as input (figure 2). Each slot was represented by six input units (one for each class). 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 that was always 1 was included to allow the network to generate initial choices. The entire input layer, thus, consisted of 73 units. Figure 2 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,

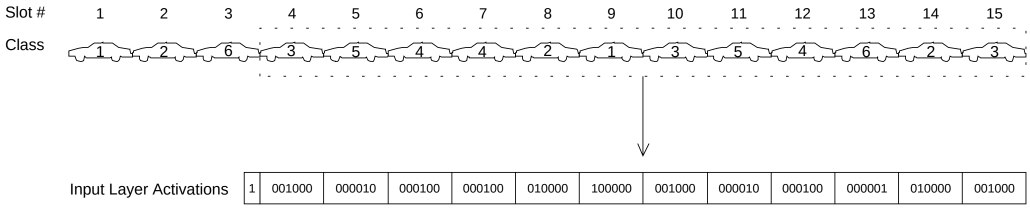

Figure 2: A partial sequence instantiation and the corresponding input to the network. The network receives the previous 12 assignments as input. For example, a car of class 3 has been assigned to slot 4. The first input unit is always 1 to allow the network to make initial choices. The next slot to be scheduled is slot 16.

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 assignment must 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 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 1995).

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 exceeded the capacity of the option station over all remaining slots, the search path was terminated.

The population consisted of 800 linear threshold neurons with each threshold set at 0. The subpopulation size ζ was 100, and 40 networks were formed per generation of neurons. Each neuron thus participated in an average of 5 networks per generation. Each neuron was encoded a 240-bit chromosome that contained ten 24-bit connection definitions.

The evaluation consisted of selecting 5 scheduling problems from a database of 1000 problem instances and using the network to order the classes in a chronological backtrack search. The problem instances contained between 10 to 25 cars, and the option requirements and station capacities were the same as in table 2. The score of each network was determined by the total number of backtracks incurred, which creates a challenging credit assignment problem for the individual value-ordering decisions.

3.3 Value-ordering 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. 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 (Kale 1990; Kumar 1992) 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.

The average number of backtracks per problem in the test set were 781 for random value ordering, 85 for the maximization of future options heuristic, and 26 for the SANE network. 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.

The SANE network appeared to take a first-fail approach to value ordering by preferring classes that place the most demand on the system. This approach is most obvious in the case of class 1, and constitutes the largest 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 them 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 considered a good problem-general approach because it directs the search toward areas in the search space with high solution densities (Kale 1990). Using a similar heuristic, Kale (1990) was able to solve an order of magnitude larger instances of the n-queens problem than with the standard left-right column ordering. In this particular case, however, SANE discovered a better ordering through domain-specific knowledge. It is the attainment of this domain-specific knowledge that separates SANE from other more problem-general approaches and should allow SANE to be effective in a broad range of problems.

4 Focusing Minimax Search

Value-ordering provided a well-studied decision task where problem-general approaches have performed poorly. SANE, however, requires no pre-existing knowledge of the decision task but instead learns its decision policies through direct interaction with the domain. In other words, since SANE can learn in domains with very sparse reinforcement, decision tasks that previously received little attention because they were either too hard to learn or analyze may be optimized by SANE networks. This section presents an application of SANE to such a novel decision task.

Almost all current game programs rely on the minimax search algorithm to return the best move. Because of time and space constraints, searching to the end of the game is not feasible for most games. Heuristic evaluation functions, therefore, are used to approximate the payoff of a state. However, heuristics create errors that propagate up the search tree, and can greatly diminish the effectiveness of minimax (Korf 1988). Minimax also does not promote risk taking, assuming that the opponent will always make the best move. Often in losing situations the best move may not be towards the highest min/max value, especially if it will still result in a loss. Knowledge of move probabilities could guide the search towards a more aggressive approach and take advantage of possible mistakes by the opponent.

Most game programs overcome weak evaluation functions by searching deeper in the tree. Presumably, as the search frontier gets closer to the goal, the evaluations become more accurate. While this may be true, there is no guarantee that deeper searches will provide frontier nodes closer to the goal states. Hansson and Mayer (1990) showed that without a sound inference mechanism, deeper searches can actually cause more error in the frontier nodes. A more directed search, therefore, seems necessary. Using SANE to focus minimax away from misinformation and towards more effective moves is an important, novel application to game playing and further demonstrates SANE's ability to incorporate domain-specific knowledge (In this case, weaknesses of minimax and the evaluation function) to form an effective decision strategy.

4.1 Creating a Focus Window

In earlier work (Moriarty and Miikkulainen 1994c), we showed how standard neuro-evolution methods can evolve a focus network in the game of Othello to decide which moves in a given board situation are to be explored. In this paper, the SANE method was applied to the same task with significantly stronger results. In the search focus task, the network sees the updated board and evaluates each move at each level of minimax search. Only those moves that are better than a threshold value will be further explored. The search continues until a fixed depth bound is reached. An evaluation function is applied to the leaf states, and the values are propagated up the tree using the standard minimax method. The α-β pruning algorithm is used as in a full-width search to prune irrelevant states.

Restricting the number of moves explored has two key advantages: (1) the branching factor is reduced, which greatly speeds up the search. As a result, searches can proceed deeper on more promising paths. (2) The focus networks are forced to decide which moves the minimax search should evaluate, and in order to play well, they must develop an understanding of the minimax algorithm. It is possible that they will also discover limitations of minimax and the evaluation function, and learn to compensate by not allowing minimax to evaluate certain moves.

Figure 3 illustrates the focused search process. The player has a choice of 5 moves (a through e) and is searching 2 moves ahead. The leaf states are evaluated according to a static evaluation function. The actual payoff value of each leaf is shown below the depth bound. The difference between these values is the error or misinformation generated by the evaluation function. Move e is the strongest move since it will generate an actual payoff of

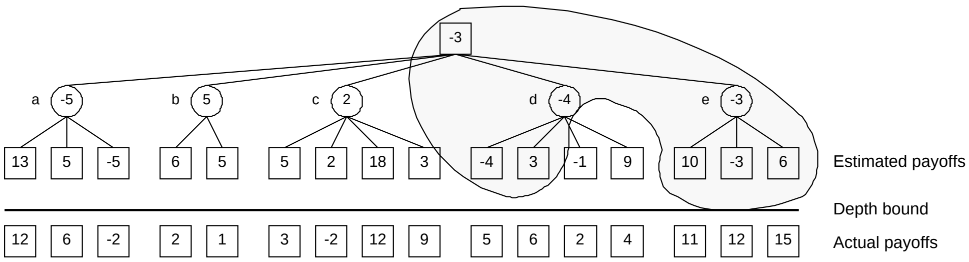

Figure 3: A minimax search to level 2. Min (circles) selects the lowest payoff and max (squares) the highest of min's choices. A full-width search considers all possible moves, while a focused search considers only those moves in the focus window (shown by the shaded region). The value for the root node is shown for a focused search. The full-width search would select move *b*, although move *e* is max's best choice. The focused search does not consider move *b* and selects move *e*.

at least 11. Because of the misinformation, which evaluates move *e* at -3, a full-width search would select move *b*, since it returns a payoff of at least 5. The focused search, however, is able to select move *e* by excluding move *b* from consideration.

4.2 Implementation of Focus Networks in Othello

Two input units were used to represent the type of piece in each board space (figure 4). If the space contains the network's piece, the first input unit is turned on (value = 1). If the space contains the opponent's piece, the second input unit is turned on. If the space is empty, neither input unit is activated. The two input units are never both on.

Each output unit corresponded directly to a space on the board. The activation of an output unit determined whether a move was to be considered or not. If the activation was greater than or equal to 0, the move was included in the focus window. Separate output units were used for the two players to allow offensive and defensive strategies to develop. Thus, the ranking for the network's moves may differ from the ranking of the opponent's moves.

To evaluate a network, it was inserted into an α-β search program and played against a full-width, fixed-depth minimax-α-β search. The number of wins over ten games played determined the network's score. To create different games, an initial state was selected randomly among the 244 possible board positions after four moves. Both players were allowed to search through the second level and used the evaluation function from the Bill program (Lee and Mahajan 1990), which is composed of large Bayes-optimized lookup tables gathered from expert games.[1] Bill was at one time the world-champion program and is still believed to be one of the best in the world. Any improvement over the current Bill evaluation function would thus be a significant result.

[1] Thanks to Richard Korf and Kai-Fu Lee for providing Bill's evaluation function.

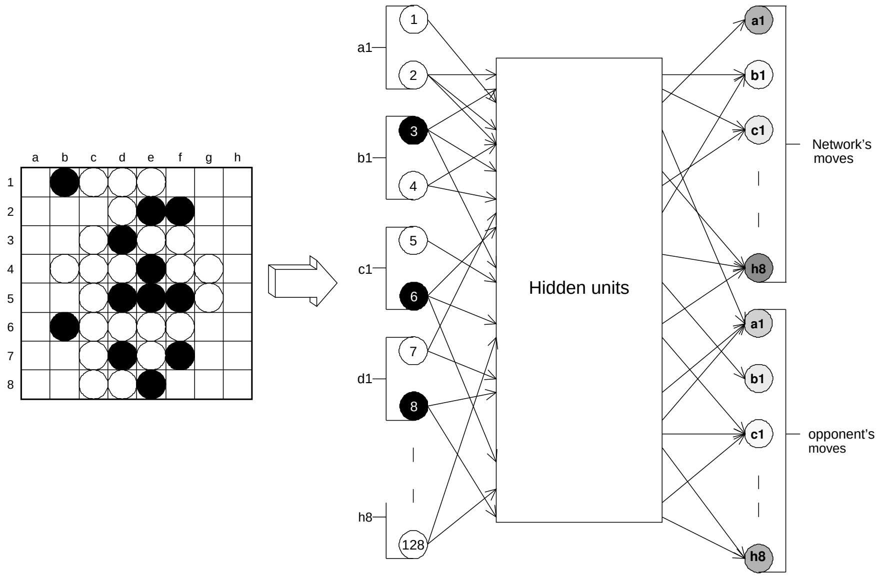

Figure 4: The architecture of the focus networks for Othello. Two inputs are used to encode each position on the board. The encoding of the first four spaces (a1, b1, c1, d1) for the given board with the network playing black are shown in the input layer. The activation of the output layer is shown by the shading. The corners (such as a1 and h8) have high activations since corners are almost always good moves.

4.3 Focused Search Results

The SANE neurons were evolved for 200 generations, which took about 11 hours of CPU time on an IBM RS6000 25T. After evolution, the best network of each generation was tested against another full-width search in each of the 244 initial games, however, this time each player was allowed to search through level 3. The network with the highest winning percentage over these tests was selected as the best network.

Table 3 shows the best focus network's performance over various search levels against the full-width opponent. The results show that the focus network was playing a comparable and in most cases better game than Bill. Most remarkably, the focus networks won while looking at only a subset of the states as the full-width search. Of all available moves to level 6, only 77% were included in the focus network's window. Since the full-width search is looking at the same moves as the focused search plus additional moves, there must be some misinformation in the additional moves that are causing it to select poor moves. Since the focused search employs the same evaluation function to the same depth and yet is selecting better moves, it appears that the focus network is shielding the root from this misinformation.

The results indicate that SANE can evolve better and more efficient game play through more selective search. SANE is able to tailor the minimax search to make the best use out of the information the evaluation function provides. SANE can optimize even highly sophisticated evaluation functions, like that of the Bill program. This is a significant improvement over the standard neuro-evolution approach used in (Moriarty and Miikkulainen 1994c), which could improve play with a weak heuristic, but could not extend to Bill's evaluation

Level	1	2	3	4	5	6
% of games won by SANE	54	54	62	49	53	51
Avg. states for SANE	198	931	5808	30964	166911	939999
Avg. states for full-width	207	977	6305	35932	212285	1217801

Table 3: The winning percentage of SANE and the average number of states examined per game for each depth bound.

function in deeper searches.

More generally, the simulations demonstrate SANE's ability to form effective decision policies in novel decision tasks. Whereas most research has improved game-playing through optimization of the evaluation function (Hansson and Mayer 1990; Lee and Mahajan 1990) or altering the minimax algorithm (Korf and Chickering 1994; McAllester 1988), SANE attacks misinformation by making search-level decisions that can overcome deficiencies in minimax and the evaluation function. Such a novel approach to game-tree search, which is one of the most-studied fields in artificial intelligence, illustrates how SANE, through its generality and ability to learn with sparse reinforcement, may uncover previously unrealizable sequential decision tasks.

5 Future Work

Each of the decision tasks described in this paper were Markovian decision problems, where the future behavior of the system depended only on the current state and future inputs. While many interesting tasks can be formulated as Markov problems, in many real-world applications, factors outside the current observable state may also influence system behavior. For example in chess, move decisions are often based not only on the current board configuration, but also on the opponent's apparent strategy up to that point. SANE can be applied to such tasks with the addition of recurrent connections. A recurrent network maintains an internal representation in its hidden layer and it can be used to identify historical factors that affect system performance. Since forming recurrent connections requires little additional computation in SANE, SANE should be able to efficiently evolve recurrent networks for non-Markovian tasks.

Future work on SANE also includes applying it to larger real-world domains with multiple decision tasks. One such domain is local area networks (LAN), where possible tasks include packet routing, resource allocation, congestion control, and priority queueing. While standard methods for LAN control exist, SANE's domain-specific knowledge, attained through direct interaction with the LAN, should produce more effective decision policies. Other possible domains include elevator control, air and automobile traffic control, and robot control. Since SANE makes few domain assumptions, it should be applicable in each of these domains as well as many others.

An important question to be explored in future research is: Can SANE simultaneously evolve networks for separate decision tasks? For example, can neurons involved in priority queueing be evolved with neurons involved in packet routing? Evolving neurons for many different networks at once should not be very different from evolving them for a single network, because in any case SANE must develop neurons that specialize and serve very different roles. The input layers and output layers of each network could be concatenated to form a single, multi-task network. Since a hidden neuron can establish connections to any input or output unit, it can specialize its connections to a single decision task or form connections between sub-networks that perform different tasks. Such inter-network connections could produce interesting interactions between decision strategies, which is an issue that to our knowledge has not been studied before.

6 Conclusion

SANE provides a powerful new mechanism for learning and performing sequential decision tasks. SANE's neural networks provide effective pattern recognition and generalization, while the genetic algorithm allows SANE to learn under sparse reinforcement. The value ordering and minimax tasks illustrate how SANE can outperform problem-general heuristics by incorporating domain-specific information such as the option capacities of an assembly line or the weaknesses of an evaluation function. SANE's few domain assumptions and ability to learn in tasks with sparse reinforcement should make it applicable to a broad range of decision tasks, including real-world problems.

References

Anderson, C. W. (1989). Learning to control an inverted pendulum using neural networks. *IEEE Control Systems Magazine*, 9:31–37.

Barto, A. G., Sutton, R. S., and Anderson, C. W. (1983). Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-13:834–846.

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.

Belew, R. K., McInerney, J., and Schraudolph, N. N. (1991). Evolving networks: Using genetic algorithm with connectionist learning. In Farmer, J. D., Langton, C., Rasmussen, S., and Taylor, C., editors, *Artificial Life II*. 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.

Hansson, O., and Mayer, A. (1990). Probabilistic heuristic estimates. *Annals of Mathematics and Artificial Intelligence*, 2:209–220.

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.

Korf, R. E. (1988). Search: A survey of recent results. In Shrobe, H. E., editor, *Exploring Artificial Intelligence*. San Mateo, California: Morgan Kaufmann.

Korf, R. E., and Chickering, D. M. (1994). Best-first minimax search: Othello results. In *AAAI-94*.

Koza, J. R., and Rice, J. P. (1991). Genetic generalization of both the weights and architecture for a neural network. In *International Joint Conference on Neural Networks*, vol. 2, 397–404. New York, NY: IEEE.

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

Lee, K.-F., and Mahajan, S. (1990). The development of a world class Othello program. *Artificial Intelligence*, 43:21–36.

Littman, M. L., and Boyan, J. A. (1993). A distributed reinforcement learning scheme for network routing. Technical Report CMU-CS-93-165, School of Computer Science, Carnegie Mellon University.

McAllester, D. A. (1988). Conspiracy numbers for min-max search. *Artificial Intelligence*, 35:287–310.

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

Moriarty, D. E., and Miikkulainen, R. (1994b). Evolutionary neural networks for value ordering in constraint satisfaction problems. Technical Report AI94-218, Department of Computer Sciences, The University of Texas at Austin.

Moriarty, D. E., and Miikkulainen, R. (1994c). 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. (1995). Discovering complex Othello strategies through evolutionary neural networks. *Connection Science*, 7(3). (To Appear).

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

Whitley, D., Dominic, S., Das, R., and Anderson, C. W. (1993). Genetic reinforcement learning for neurocontrol problems. *Machine Learning*, 13:259–284.