Evolutionary Computation, 5(4), 1998. (In Press).

Forming Neural Networks through Efficient and Adaptive Coevolution*

David E. Moriarty
Information Sciences Institute
University of Southern California
4676 Admiralty Way
Marina del Rey, CA 90292
moriarty@isi.edu

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

Abstract

This article demonstrates the advantages of a cooperative, coevolutionary search in difficult control problems. The SANE system coevolves a population of neurons that cooperate to form a functioning neural network. In this process, neurons assume different but overlapping roles, resulting in a robust encoding of control behavior. SANE is shown to be more efficient, more adaptive, and maintain higher levels of diversity than the more common network-based population approaches. Further empirical studies illustrate the emergent neuron specializations and the different roles the neurons assume in the population.

1 Introduction

Artificial evolution has become an increasingly popular method for forming control policies in difficult decision problems (Grefenstette, Ramsey, & Schultz, 1990; Moriarty & Miikkulainen, 1996a; Whitley, Dominic, Das, & Anderson, 1993). Such applications are very different from the function optimization tasks to which evolutionary algorithms (EA) have been traditionally applied. For example, it is no longer desirable to converge the population to the best solution, since convergence will hinder adaptation of the population in dynamic environments. Also, the large number of fitness evaluations that are freely granted to the standard EA in toy domains are not available in real world decision problems. It is becoming clear that for complex structures such as neural network controllers, a different kind of evolutionary algorithm is necessary.

Cooperative Coevolutionary algorithms (Horn, Goldberg, & Deb, 1994; Moriarty & Miikkulainen, 1996a; Paredis, 1995; Potter, De Jong, & Grefenstette, 1995; Smith, Forrest, & Perelson, 1993; Whitehead & Choate, 1995) offer a promising alternative to the canonical EA in difficult and dynamic problems. The key difference in the cooperative coevolutionary model compared to the standard model is that each individual represents only a partial solution to the problem. Complete solutions are formed by grouping several (or all) individuals together. The goal of an individual is

*This research was performed at The University of Texas at Austin and was supported in part by the National Science Foundation under grant #IRI-9504317.

thus to optimize one piece of the solution and cooperate with other partial solutions that optimize other pieces. The hypothesis is that several parallel searches for different pieces of the solution is more efficient than a single search for the entire solution. Moreover, by maintaining the different solution pieces in a single population, the population does not converge to a single individual. Diversity is maintained and the EA can utilize its recombination operators throughout evolution.

This article demonstrates the advantages of cooperative, coevolutionary algorithms in difficult control problems using a new system called SANE (Symbiotic, Adaptive Neuro-Evolution). SANE was designed as an efficient method for building neural networks in dynamic environments (Moriarty & Miikkulainen, 1996a). Unlike most neuro-evolutionary approaches, which operate on a population of neural networks, SANE evolves a *population of neurons*. Each neuron's task involves establishing connections with other neurons in the population to form a functioning neural network. SANE's performance improvements over the more standard EA in neural network learning are twofold. First, since SANE recognizes neurons as the functional components of neural networks, it can more accurately search and evaluate the genetic building blocks. Second, since no one neuron can perform well alone, evolutionary pressures exists to evolve several different neuron types or *specializations*. Thus, SANE maintains diverse populations.

The body of this article is organized as follows. The next section gives some background and motivates SANE's coevolutionary approach to neuro-evolution. Section 3 gives the implementation details of SANE. Section 4 empirically evaluates SANE by comparing it with more standard approaches to neuro-evolution in a mobile robotics task. Section 5 illustrates the specializations within SANE's population and uncovers some of the different roles that the neurons assume in the networks. Work related to SANE is presented in section 6 and section 7 describes some successful applications. We outline some future directions of symbiotic evolution in section 8 and draw our conclusions in the final section.

2 Importance of Diversity

To find the best combination of genetic building blocks, evolutionary algorithms continually select and breed the best individuals in the population. Through this process, populations normally lose diversity and eventually converge around a single "type" of individual (Goldberg, 1989). Such convergence is undesirable for two reasons: (1) populations often converge on suboptimal peaks and (2) converged populations cannot adapt well to changes in the task environment. While these two problems may have limited effect in standard function optimization, they have very large consequences when evolving decision strategies in complex and dynamic domains.

An evolutionary algorithm in a converged population can normally only proceed by randomly mutating the single solution representation, which produces a very slow and inefficient search. A genetic search with a diverse population, however, can continue to utilize recombination to generate new structures and make larger traversals of the solution space in shorter periods of time. In many complex problems, search efficiency is paramount for generating effective decision policies in a timely manner. Diversity is equally important when changes occur in the domain. Often in decision tasks in dynamic or unstable environments, policies must be quickly revised to avoid costly effects. A diverse population can more adeptly make the modifications necessary to compensate for the domain changes.

It is clear that a convergent evolutionary algorithm is not the best kind of search strategy for difficult decision and control problems. Maintaining diverse populations, however, is very difficult

and remains an open research issue in the evolutionary algorithms community. The most common method is to use a less aggressive genetic selection strategy or a high rate of mutation. Weak selection strategies do not ensure diversity, but rather slow evolution and delay the convergence of the population. Slower evolution can help prevent premature convergence, but often at the expense of slower searches. Section 4 will present experiments that demonstrate this phenomenon. The second strategy, increasing the mutation rate, only artificially injects diversity into the population through noise. Despite their obvious disadvantages, these two methods generally produce better search behavior than an aggressive, convergent EA, and their adoption has become commonplace. The SANE system will demonstrate that aggressive selection and recombination strategies can work well if tempered with effective diversity pressures.

Several more intelligent methods have been developed to enforce population diversity, including fitness sharing (Goldberg & Richardson, 1987), crowding (De Jong, 1975), and local mating (Collins & Jefferson, 1991). Each of these techniques relies on external genetic functions that prevent convergence of the genetic material. The diversity assurances, however, are normally achieved through very expensive operations. For example, in Goldberg's fitness sharing model, similar individuals are forced to share a large portion of a single fitness value from the shared solution point. Sharing decreases the fitness of similar individuals and causes evolution to select against individuals in overpopulated niches. While fitness sharing is effective at maintaining diversity, it incurs a heavy computational expense. Sharing requires $O(n^2)$ similarity comparisons each generation, where n is the size of the population. In large populations with large chromosomes, comparison-based diversity methods such as sharing, crowding, and local mating are simply not practical (Smith et al., 1993).

A more recent technique for ensuring diversity has been termed *implicit fitness sharing* (Horn et al., 1994; Smith et al., 1993). No comparisons are made between individuals. Instead, diversity pressures are built into the task through cooperative behavior among the individuals in the population. The individuals no longer represent complete solutions to the problem, but rather represent only partial solutions and must cooperate with other individuals to form a full solution. By reducing the capacity of individuals and *coevolving* them together, evolution searches for several different types of individuals that together solve the problem. Implicit fitness sharing also presents a very nice side effect. While evolution searches for individuals that optimize different aspects of the problem, it performs several parallel searches in decompositions of the solution space, which can greatly speed up evolution.

The core evolutionary strategy of the SANE neuro-evolution system, presented in the next section, borrows many ideas from the implicit fitness sharing models. Specifically, the notion of cooperating individuals that represent only partial solutions is incorporated in SANE to promote diversity and search efficiency. Several modifications, however, were necessary to tailor implicit fitness sharing to evolving neural networks. These changes are highlighted in the next section and in the related work section.

3 Symbiotic Adaptive Neuro-Evolution

3.1 Evolving Symbiotic Neurons

In almost all approaches to neuro-evolution, each individual in the population represents a complete neural network that is evaluated independently of other networks in the population (Belew, McInerney, & Schraudolph, 1991; Koza & Rice, 1991; Nolfi & Parisi, 1992; Whitley et al., 1993).

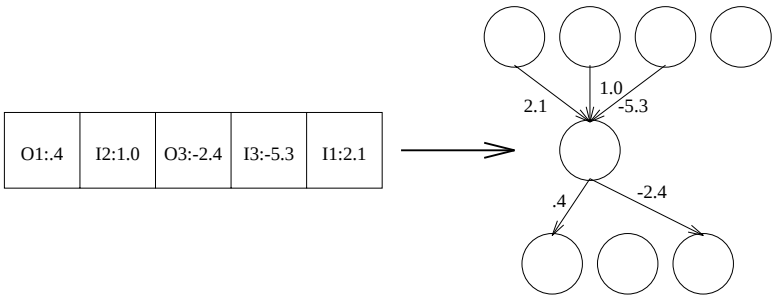

Figure 1: An individual in SANE's population and the hidden neuron it defines. Hidden neurons are defined by a series of weighted connections to be made from the input layer or to the output layer. For example, the first gene specifies a connection to the first output unit with a weight of 0.4. The encoding shown is a simplified form of SANE's actual neuron encoding, which is described in section 3.3.

As described in the previous section, by treating each member as a separate, full solution, the evolutionary algorithm focuses the search towards a single dominant individual. Such concentration can greatly impede search progress in both complex and dynamic tasks. In contrast, the SANE method restricts the scope of each individual to a single neuron. More specifically, each individual represents a hidden neuron in a 2-layer neural network (figure 1). In SANE, complete neural networks are built by combining several neurons. Figure 2 illustrates the difference between standard neuro-evolution and the neuro-evolution performed in SANE.

Since no single neuron can perform the whole task alone, the neurons must optimize one aspect of the neural network and connect with other neurons that optimize other aspects. Evolutionary pressures therefore exist to evolve several different types or *specializations* of neurons.[1] In this way, the neurons will form a *symbiotic* relationship. It follows that the evolution performed in SANE can be characterized as *symbiotic evolution*. We define symbiotic evolution as a type of coevolution where individuals explicitly cooperate with each other and rely on the presence of other individuals for survival. Symbiotic evolution is distinct from most coevolutionary methods, where individuals compete rather than cooperate to survive. A more detailed discussion of the relationship between symbiotic and competitive coevolution is presented in the related work section.

The advantages of symbiotic evolution are twofold. First, the neuron specializations ensure diversity which discourages convergence of the population. A single neuron cannot "take over" a population since to achieve high fitness values, there must be other specializations present. If a specialization becomes too prevalent, its members will not always be combined with other specializations in the population. Thus, redundant partial solutions do not always receive the benefit of other specializations and will incur lower fitness evaluations. Evolutionary pressures are therefore present to select against members of dominant specializations. This is quite different from standard evolutionary approaches, which always converge the population, hopefully at the global optimum, but often at a local one. In symbiotic evolution, solutions are found in diverse, *unconverged* populations. By maintaining diverse populations, SANE can continue to use its recombination operators to build effective neural structures.

In addition to maintaining diverse populations, evolution at the neuron level more accurately evaluates the genetic building blocks. In a network-level evolution, each neuron is implemented only

[1] The term specialization is used rather than species since each neuron does not represent a full solution to the problem. Additionally, biological species do not inter-breed, whereas SANE's specializations do.

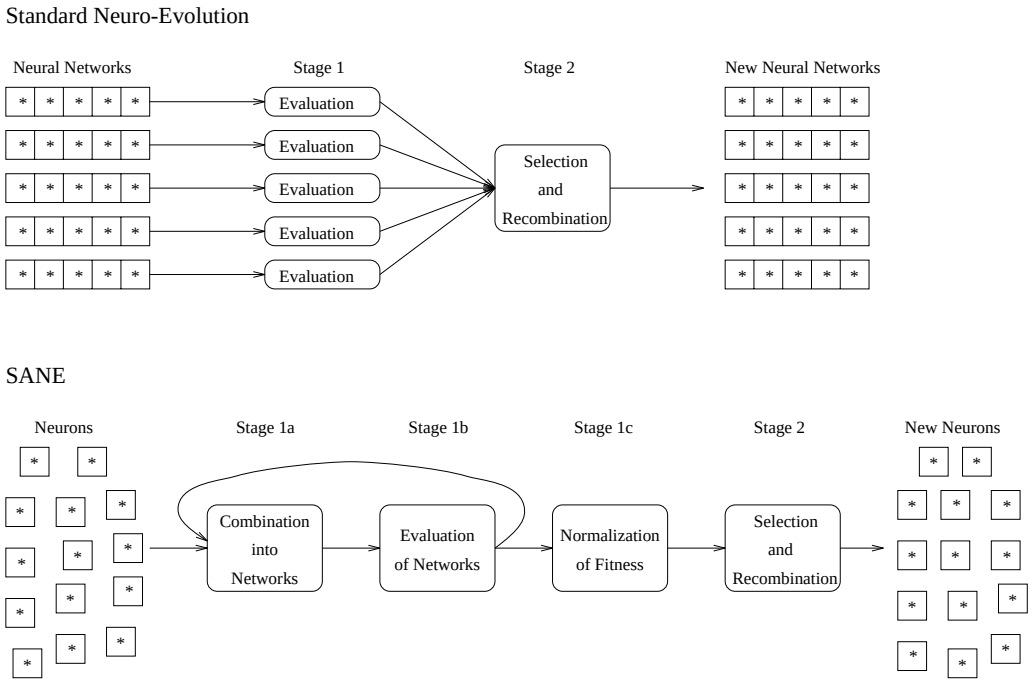

Figure 2: An illustration of the neuro-evolution performed in SANE compared to the standard approach to neuro-evolution. The standard approach maintains a population of neural networks and evaluates each independently. SANE maintains a population of neurons and evaluates each in conjunction with other neurons. Step 1 (the evaluation step) in SANE is broken into three substeps. Neurons are continually combined with each other and the resulting networks are evaluated in the task. Each neuron receives a normalized fitness based on the performance the networks in which it participates.

with the other neurons encoded on the same chromosome (e.g., figure 2). With such a representation, a very good neuron may exist on a chromosome but be subsequently lost because the other neurons on the chromosome are poor. In a neuron-level evolution, neurons are continually recombined with many different neurons in the population, which produces a more accurate evaluation of the neural network building blocks.

Essentially, a neuron-level evolution takes advantage of the *a priori* knowledge that individual neurons constitute basic components of neural networks. A neuron-level evolution *explicitly* promotes genetic building blocks in the population that may be useful in building other networks. A network-level evolution does so only *implicitly*, along with various other sub- and superstructures (Goldberg, 1989). In other words, by evolving at the neuron level the evolutionary algorithm is no longer relied upon to identify neurons as important building blocks, since neurons are the object of evolution.

3.2 Maintaining Effective Neuron Collections

Neuron evolution alone, however, is not sufficiently powerful to generate the complex networks necessary in difficult tasks. Knowledge of the useful combinations of neurons must be maintained and exploited. Combining neurons without such intelligent direction is undesirable for two reasons. First, the neurons may not be combined with neurons that work well together. Thus, a very good

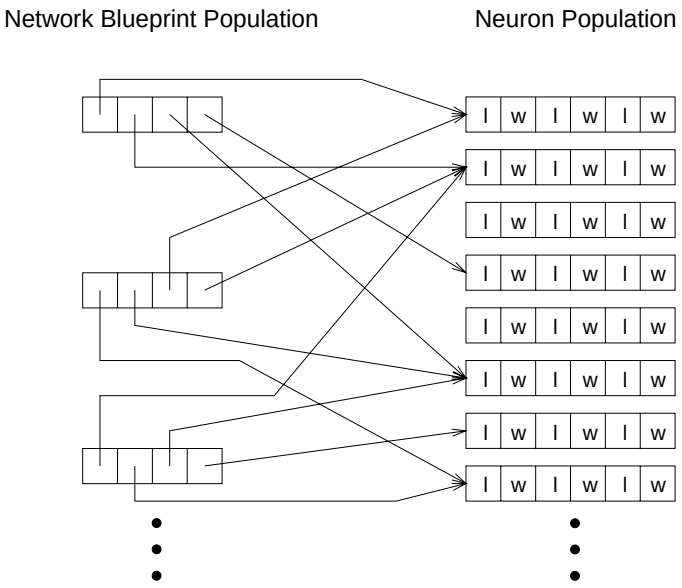

Figure 3: An overview of the network blueprint population in relation to the neuron population. Each member of the neuron population specifies a series of connections (labels and weights) to be made from the input layer or to the output layer within a neural network. Each member of the blueprint population specifies a series of pointers to specific neurons which are used to build a neural network. The neuron population searches for effective partial networks, while the blueprint population searches for effective combinations of partial networks.

neuron may be lost, because it was ineffectively combined during a generation. The second problem is that the quality of the networks varies greatly throughout evolution. In early generations this works as an advantage, since the search produces many different types of networks to find the most effective neurons. However, in later generations, when the search should focus on the best networks, the inconsistent networks often stall the search and prevent the global optima from being located. Experiments in section 4.3 will demonstrate this phenomenon.

An outer loop mechanism is necessary to maintain knowledge of the good neuron combinations. Many different approaches could perform the necessary record keeping ranging from maintaining complex tables which record the fitness of each neuron when combined with other neurons to simply remembering the top neuron combinations of the previous generation. Clearly, the memory requirements of the first method make it impractical. For example, if 8 neurons were used to build a network from a population of 800 neurons, a complete record of all neuron combinations would contain 800^8 entries. Conversely, the second method maintains very little information of the history of each neuron and may provide only limited benefit. An intermediate solution seems appropriate.

The current method of maintaining useful neuron combinations in SANE is to evolve a layer of neural network records or *blueprints* on top of the neuron evolution. The blueprint population maintains a record of the most effective neuron combinations found with the current population of neurons and uses this knowledge as the basis to form the neural networks in the next generation. Figure 3 shows the relationship between the blueprint and neuron populations. Each blueprint specifies a collection of neurons that have performed well together.

Maintaining network blueprints produces more accurate neuron evaluations and concentrates the search on the best neural networks. Since neurons are systematically connected based on past

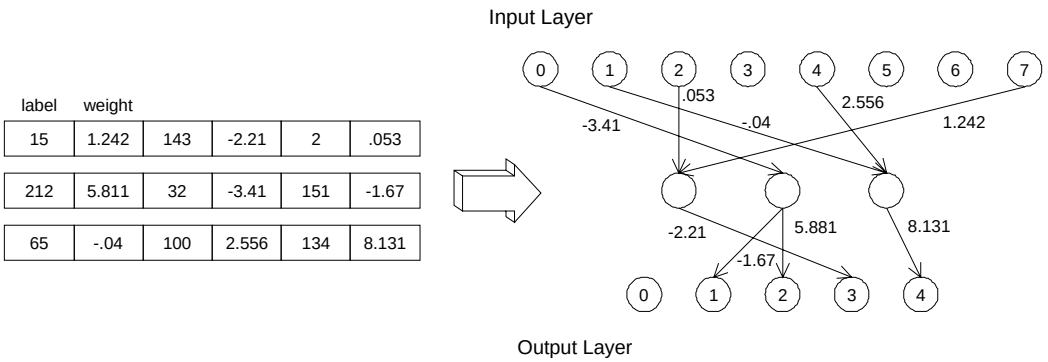

Figure 4: Forming an 8 input, 3 hidden, 5 output unit neural network from three hidden neuron definitions. The chromosomes of the hidden neurons are shown to the left and the corresponding network to the right. In this example, each hidden neuron has 3 connections.

performance, they are more consistently combined with other neurons that perform well together. Additionally, better-performing neurons garner more pointers from the blueprint population and thus participate in a greater number of networks. Biasing the neuron participation towards the historically better-performing neurons provides more accurate evaluations of the top neurons. The sacrifice, however, is that newer neurons may not receive enough trials to be accurately evaluated. In practice, allocating more trials to the top neurons produces a significant improvement over uniform neuron participation.

The primary advantage of evolving network blueprints, however, is the exploitation of the best networks found during evolution. By *evolving* the blueprint population, the best neuron combinations are also recombined to form new, potentially better, collections of neurons. The blueprint level evolution thus provides a very exploitive search that can build upon the best networks found during evolution and focus the search in later generations.

3.3 SANE Implementation

SANE[2] maintains and evolves two populations: a population of neurons and a population of network blueprints. Each individual in the neuron population specifies a set of connections to be made within a neural network. Each individual in the network blueprint population specifies a set of neurons to include in a neural network. Conjunctively, the neuron evolution searches for effective partial networks, while the blueprint evolution searches for effective combinations of the partial networks.

Each individual in the neuron population represents a hidden neuron in a 2-layer feed-forward network. Neurons are defined in bitwise chromosomes that encode a series of connection definitions, each consisting of an 8-bit label field and a 16-bit weight field. The absolute 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 4 shows how a neural network is formed from three sample hidden neuron definitions.

[2]The source code can be obtained from the UTCS Neural Networks' home page: http://www.cs.utexas.edu/users/nn/.

1.	Clear fitness value of each neuron and blueprint.
2.	Select ζ neurons from the population using a blueprint.
3.	Create a neural network from the selected neurons.
4.	Evaluate the network in the given task.
5.	Assign the blueprint the evaluation of the network as its fitness.
5.	Repeat steps 2-4 for each individual in the blueprint population.
7.	Assign each neuron the evaluation of the best 5 networks in which it participated.
8.	Perform crossover and mutation operations on the both populations.

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

Each individual in the blueprint population contains a series of neuron pointers. More specifically, a blueprint chromosome is an array, of size ζ, of address pointers to neuron structures. Figure 3 illustrates how the blueprint population is integrated with the neuron population. Initially, the chromosome pointers are randomly assigned to neurons in the neuron population. During the neuron evaluation stage, subpopulations of neurons are selected based on each blueprint's array of pointers.

Since SANE operates on bit strings and uses both mutation and recombination, its search strategy fall under the *genetic algorithm* (Holland, 1975; Goldberg, 1989) method of evolutionary computation. SANE uses a generational genetic algorithm which operates over two main phases: evaluation and recombination. In the evaluation phase, the goal is to evaluate the fitness of each neuron and network blueprint in the populations. Blueprints are evaluated based on the performance of the neural network that they specify. Neurons are evaluated through combinations with other neurons.

The basic steps in the evaluation phase are as follows (listed table 1): during the evaluation stage, each blueprint is used to select neuron subpopulations of size ζ to form a neural network. Each blueprint receives the fitness evaluation of the resulting network and each neuron receives the summed fitness evaluations of the *best five* networks in which it participated. Calculating fitness from the best five networks, as opposed to all of the neuron's networks, discourages selection against neurons that are crucial in the best networks, but ineffective in poor networks. For example in a robot arm manipulation task, a neuron that specializes in small movements near the target would be effective in networks that position the arm close to the target, but useless in networks that do not get anywhere near the target. Such neurons are very important to the population and should not be penalized for the poor networks that they cannot help.

After the evaluation stage, the neuron population is ranked based on the fitness values. For each neuron in the top 25% of the population, a mate is selected randomly among the top 25%. Each mating operation creates two offspring: a child created through a one-point crossover operation and a copy of one of the parent chromosomes. In SANE, one of the offspring produced by crossover is chosen at random to enter the population. Copying one of the parents as the second offspring reduces the effect of adverse neuron mutation on the blueprint-level evolution. This effect will be further explained in the context of the blueprint evolution later in this section. The two offspring replace the worst-performing neurons (according to rank) in the population. In each generation, 50% of the population is replaced by new offspring. Mutation at the rate of 0.1% per bit position is performed on the entire population as the last step in each generation.

Such an aggressive, elitist breeding strategy is not normally used in evolutionary applications,

since it leads to quick convergence of the population. SANE's neuron evolution, however, performs quite well with such an aggressive selection strategy, since it contains strong evolutionary pressures against convergence.

In the blueprint population, since the chromosomes are made up of address pointers instead of bits, crossover only occurs between pointers. The new offspring receive the same address pointers that the parent chromosomes contained. In other words, if a parent chromosome contains a pointer to a specific neuron, one of its offspring will point to that same neuron (barring mutation). The current evolutionary algorithm on the blueprint level is identical to the aggressive strategy used at the neuron level, however the similarity is not essential and a more-standard evolutionary algorithm or other methods of evolutionary computation could be used. Empirically, the aggressive strategy at the blueprint level coupled with the strong mutation strategy described below, has outperformed many of the more-standard evolutionary algorithms.

To avoid convergence problems at the blueprint level, a two-component mutation strategy is employed. First, a pointer in each offspring blueprint is randomly reassigned to another member of the neuron population at a rate of 1%. This strategy promotes participation of neurons other than the top neurons in subsequent networks. Thus, a neuron that does not participate in any networks can acquire a pointer and participate in the next generation. Since the mutation only occurs in the blueprint offspring, the neuron pointers in the top blueprints are always preserved.

The second mutation component is a selective strategy designed to take advantage of the new structures created by the neuron evolution. Recall that a breeding neuron produces two offspring: a copy of itself and the result of a crossover operation with another breeding neuron. Each neuron offspring is thus similar to and potentially better than its parent neurons. The blueprint evolution can use this knowledge by occasionally replacing pointers to breeding neurons with pointers to offspring neurons. In the experiments described in this paper, pointers are switched from breeding neurons to one of their offspring with a 50% probability. Again, this mutation is only performed in the offspring blueprints, and the pointers in the top blueprints are preserved.

The selective mutation mechanism described above has two advantages. First, because pointers are reassigned to neuron offspring that are the result of crossover, the blueprint evolution can explore new neuron structures. Second, because pointers are also reassigned to offspring that were formed by copying the parent, the blueprints become more resilient to adverse mutation in the neuron evolution. If pointers were not reassigned to copies, many blueprints would point to the same exact neuron, and any mutation to that neuron would affect every blueprint pointing to it. When pointers are occasionally reassigned to copies, however, such mutation is limited to only a few blueprints. The effect is similar to schema promotion in standard evolutionary algorithms. As the population evolves, highly fit schema (i.e. neurons in this case) become more prevalent in the population, and mutations to one copy of the schema do not affect other copies in the population.

4 Performance Evaluation

The coevolutionary search in SANE is quite different from those of standard neuro-evolution, and it is important to evaluate the performance improvements. This section empirically evaluates SANE through comparisons with more standard neuro-evolutionary approaches. The experiments show the advantages of SANE's symbiotic search strategy in terms of search efficiency, diversity, and adaptability.

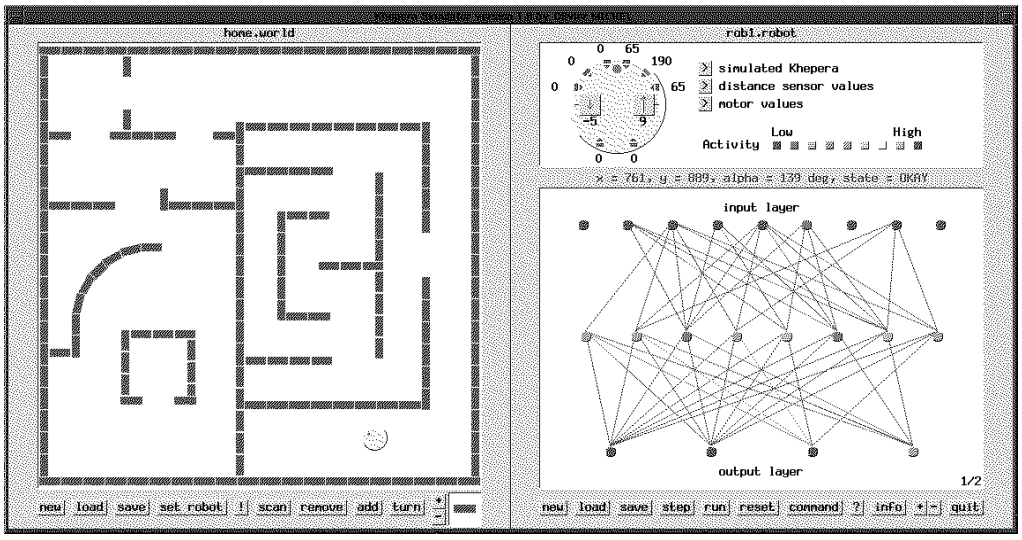

Figure 5: The interface to the Khepera 1.0 simulator (Michel 1995). The window shows a view of Khepera and the configuration of its artificial world.

4.1 The Khepera Robot Simulator

The domain chosen for evaluation of SANE was mobile robotics, or more specifically, controlling the Khepera mobile robot (Mondada, Franzi, & Ienne, 1993).[3] Michel (1995) has developed a simulator of the Khepera robot, which contains useful X window utilities for visualizing neural network controllers. Network architectures and activations can be viewed during the simulation along with the activation of the robot's sensors and motors. These utilities, along with the real world sensory input and motor output, make the Khepera simulator an excellent utility to evaluate many of the features and components of SANE.

Khepera is a tiny robot (5 cm diameter) designed for research and teaching purposes. Khepera contains both infrared and light sensors positioned around its circumference. Despite its size, the robot is not easy to control. Khepera provides real world sensory information and requires a strong grounding to the motor outputs to effectively maneuver the robot (Mondada et al., 1993). The I/O resources of the simulator were designed to accurately reflect those of the real robot. The eight infrared sensors detect the proximity of objects by light reflection and return values between 0 and 1023 depending on the color level. A value of 0 indicates that no object is sensed, and a value of 1023 indicates an object almost touching the sensor. Khepera has two wheels, controlled by two separate motors, which can receive speed ranges from -10 to 10.

Figure 5 shows a snapshot of the simulator window and the layout of Khepera's world.[4] The Khepera robot was placed in the world with the following goal: within a specific allotted time, move as far away (in Euclidean distance) from your starting position as possible without colliding with obstacles. Thus, an effective controller must accurately translate the infrared sensory information into specific motor outputs to both move the robot forward (or backward) and maneuver the robot around obstacles.

[3] Information on Khepera can be found at http://wwwi3s.unice.fr/ om/khep-sim.html.

[4] The specific world that was used was the lab0.world from the Khepera 1.0 simulator package.

4.2 Experimental Setup

The first experiments were designed to compare SANE's symbiotic evolution to more standard neuro-evolution techniques. Four evolutionary approaches were tested in the Khepera simulator: (1) SANE, (2) a standard neuro-evolution approach using the same aggressive selection strategy as SANE, (3) a standard neuro-evolution approach using a less aggressive, tournament selection strategy, and (4) a version of SANE without the network blueprint population.

The standard neuro-evolution approaches evolve a population of neural networks. Each individual's chromosome consisted of 8 neuron definitions, encoded in the same fashion as in SANE. The difference in the two standard approaches is in the underlying genetic selection strategies. The first approach, *Standard Elite*, uses the same aggressive, elitist selection strategy as SANE. Thus, the only difference between SANE and standard elite is the level of evolution. SANE performs evolution on the neuron level and the standard elite approach on the network level. Comparisons of SANE to the standard elite approach demonstrate how the diversity pressures in the neuron evolution allow for aggressive searches that perform poorly in standard evolutionary algorithms because of premature convergence.

The second standard neuro-evolution approach uses a less aggressive binary tournament selection strategy called *Standard Tournament*. Two random individuals are selected from the population, and the individual with a higher fitness is used for recombination. This strategy creates a selection bias towards the top individuals but does not preclude recombination of poor individuals. Contrasted with SANE's elitist approach, where the top 10% of the population participates in over half of the recombination operations, binary tournament selection is much less aggressive. Recent research has shown tournament selection to be the preferred method of genetic selection in terms of its growth ratios for discouraging premature convergence (Goldberg & Deb, 1991). Comparisons of SANE to the standard tournament neuro-evolution approach demonstrate the performance of the symbiotic search relative to a more "state of the art" genetic search strategy.

The fourth evolutionary approach, *Neuron SANE*, is a symbiotic neuron search without the higher-level blueprint evolution. Instead of using a population of network blueprints to form the neural networks, Neuron SANE forms networks by randomly selecting subpopulations of neurons. Comparisons of SANE to Neuron SANE can thus effectively gauge the contribution of the blueprint-level evolution.

To focus the comparison on the different strategies of neuro-evolution, rather than the choice of parameter settings, several preliminary experiments were run to discover effective parameter values for each approach. Table 2 summarizes the parameter choices for each method. With the standard approaches, a population size of 100 networks was found to be more effective than populations of 50 or 200. Keeping the number of network evaluations per generation constant across each approach, a neuron population size of 800 for SANE and 200 for Neuron SANE performed well. A 0.1% mutation rate was used for all four approaches.

Neuron SANE requires a smaller population than SANE because its neurons are evaluated through random combinations. The population must be small enough to allow neurons to participate in several networks per generation. For example, randomly selecting 8 neurons for 100 networks in a 200 neuron population gives each neuron an expected network participation rate of 4 networks per generation. In SANE, the neuron population is not as restricted, since neuron participation is dictated by the network blueprints. SANE skews the participation rate towards the best neurons and leaves many neurons unevaluated in each generation. An unevaluated neuron is normally garbage, since no blueprint uses it to build a network.

	SANE	Neuron SANE	Standard Elite	Standard Tournament
Neuron Population	800	200	-	-
Network Population	100	-	100	100

Table 2: Implementation parameters for each method.

	> 100	> 150	> 200	> 250	> 300
SANE	1	6	14	26	41
Neuron SANE	1	8	14	34	64
Standard Elite	3	13	37	65	-
Standard Tournament	2	10	21	40	79

Table 3: The average number of generations to reach the desired level of distance over the 50 position test set. For example, SANE required 26 generations on average to generate a network that averaged over 250 cm of distance on the test set. SANE's evolution was the most efficient requiring half of the evaluations of the standard approaches to reach the top level of performance.

Three experiments were run to compare the four approaches: a performance analysis, a diversity analysis, and an adaptive analysis. The results of each of the experiments were averaged over 20 simulations.

4.3 Performance Analysis

The first experiment tested the learning speed and solution quality of each evolutionary approach. Populations were evolved in the Khepera simulator for 80 generations. During each network evaluation, the robot was placed in a random position in the Khepera world, and the network was allowed to move the robot until it hit an obstacle or the maximum number of moves, 200, was exhausted. The fitness of each network was the maximum Euclidean distance the robot moved from its starting position.

To generate a learning curve, the best network of each generation (according to fitness) was tested on a 50 start-position test set. The average distance on the test set defines the performance of that generation. For each generation, the learning curve plots the best performance found at or before that generation. The learning curve thus shows the quality of solution that can be expected by each generation.

Figure 6 shows the learning curve from the performance analysis and table 3 shows the average number of generations to reach the specific levels of distance over the test set. As expected, the standard approach with the elite, aggressive selection strategy performed poorly. The search was inconsistent; it either found very good networks or stalled with very poor networks, which is characteristic of an aggressive, convergent search. If the search converged on a good network, it could often tweak it with mutation into a great network. Otherwise, it remained stuck with a suboptimal solution. These experiments confirmed that for a network level evolution, tournament selection is a better selection strategy.

SANE performed the most efficient search, finding networks that averaged 300 cm of distance in half as many generations as the standard tournament search and less than half of the generations of the standard elite search. Unlike the standard elite approach, SANE's aggressive searches were very consistent. Only one of the SANE simulations out of the 20 returned a final network that averaged less than 300 cm. SANE's neuron-based searches, thus, do not appear as susceptible to

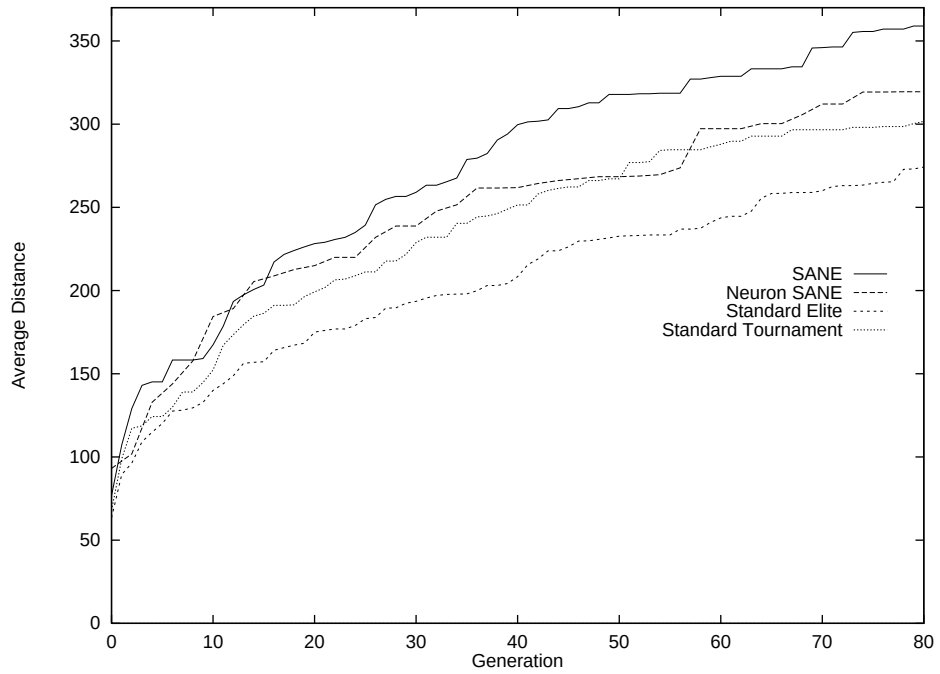

Figure 6: Comparison of the learning speeds of the different evolutionary approaches. The distance refers to the average distance over a 50 position test set for the best network found at or before each generation. The distances are averaged over 20 simulations. Simulations were run for 80 generations, because at that point the standard approaches began to plateau. The learning curve demonstrates the efficiency gain from the neuron and blueprint populations.

premature convergence as an aggressive evolutionary algorithm operating on a population of neural networks.

The neuron-only version of SANE was as efficient as SANE in early generations, but was unable to maintain the same efficiency in later generations. Without a mechanism to propagate knowledge of the good networks that are formed, it is difficult for Neuron SANE to build upon the best networks. The poor late performance gives strong evidence to this problem and effectively shows the contribution of the network blueprint evolution. Neuron SANE performed comparably to the standard tournament approach.

To show that the difference in the curves are statistically significant, we applied a single tailed t-test to find 95% confidence intervals over the differences between SANE and the other approaches. With 95% confidence, the differences in average distance after 80 generations are in the following ranges: Neuron SANE, $[39.5, 5.3]$, Standard Elite $[84.9, 51.2]$, Standard Tournament $[57.4, 20.1]$. Thus, there is a statistically significant difference ($p < .05$) between the performance of SANE after 80 generations and the other three approaches.

4.4 Diversity Analysis

The second experiment tested the diversity level of the populations throughout evolution. The goal of this test was to demonstrate the population-level differences between the coevolutionary and standard approaches. Populations were evolved for 80 generations as in the first experiment, but after each generation, the population diversity was measured. A diversity metric, Φ, can be

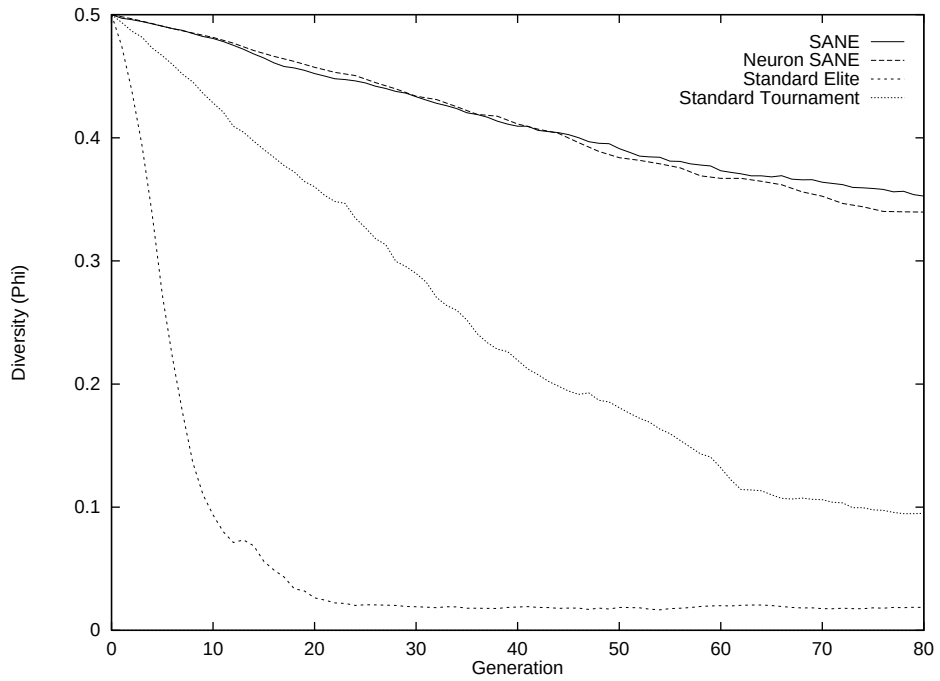

Figure 7: The population diversity for each simulation. The neuron-based approaches maintain very high levels of diversity, while the network-based approaches converge to a single solution.

generated by taking the average Hamming distance between every two chromosomes, divided by the length of the chromosome:

$$\Phi = \frac{2\sum_{i=1}^{n}\sum_{j=i+1}^{n} H_{i,j}}{n(n-1)l},$$

where n is the population size, l is the length of each chromosome, and $H_{i,j}$ is the Hamming distance between chromosomes i and j. The value Φ represents the probability that a given bit at a specific position on one chromosome is different from a bit at the same position on a different chromosome. Thus, a random population would have $\Phi = 0.5$ since there is a 50% probability that any two bits in the same position differ.

Figure 7 shows the average diversity levels at each generation. The convergence of the standard elite approach is quite dramatic. Within 10 generations, 95% of the bit positions were identical. It is this phenomenon that leads most evolutionary algorithm implementors away from aggressive selection strategies and towards more conservative approaches like tournament selection. The tournament standard approach did converge much slower, but after 60 generations 90% of its bit positions were identical as well. Both SANE and Neuron SANE maintained very diverse populations throughout evolution, which confirms our hypothesis that SANE can perform a very aggressive search while maintaining a high level of diversity. Aggression balanced with diversity is the core of SANE's search strategy and is what sets it apart from current neuro-evolution approaches. Diversity allows SANE to improve its networks in later generations and, as demonstrated in the next experiment, adapt in changing environments.

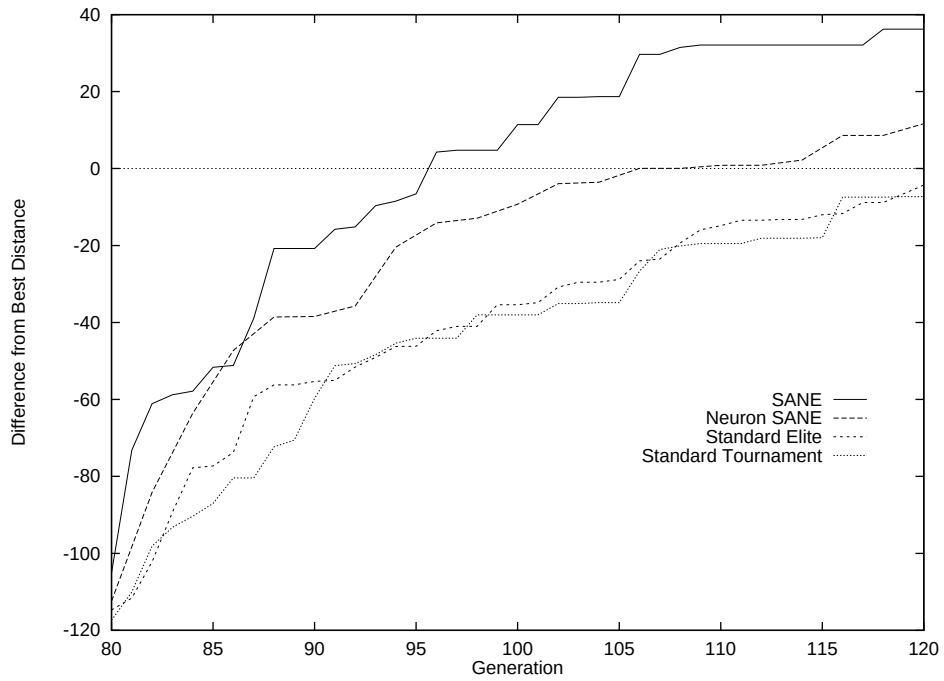

Figure 8: Adaptive comparisons. After evolving for 80 generations, a back sensor is fixed at 1.0. The graph plots the difference in performance from the best performance in the previous 80 generations.

4.5 Adaptive Analysis

The third experiment tested the ability of each approach to adapt to changes in the domain. Populations were evolved for 80 generations as in the first experiment. After 80 generations, the right back sensor of the khepera robot was "damaged", and the populations were evolved for 40 more generations. The sensor was set to a constant value of 1.0, which gives the illusion of an immediate obstacle to the rear of the robot. To adapt, the populations must learn to ignore the malfunctioning sensor and rely on the other back sensor. This experiment was designed to give a realistic situation for which adaptive behavior is necessary.

Figure 8 plots a learning curve for the simulations in the adaptive analysis. The y axis represents the difference in performance relative to the performance level achieved before the malfunctioning sensor. The horizontal line at 0 on the y axis represents the point where the search achieves the same performance as before the domain change. The relative distance is plotted rather than the absolute distance because each approach achieves a different level of performance after 80 generations.

As expected, the converged populations were much less adaptive than the diverse populations. After the sensor malfunctioned, each of the approaches lost about the same amount of performance, approximately 120 cm of distance on average. SANE quickly matched and surpassed its previous performance in an average of 15 generations. Neuron SANE required 25 generations to fully adapt. On average, neither of the standard approaches were able to achieve the performance level they had reached with the correct sensor within 40 generations. The performance of SANE and Neuron SANE demonstrates how evolution in diverse populations is capable of quickly generating new solutions when the task or environment changes. Without diversity, evolution is seriously limited and cannot generate new structures in a timely manner.

5 Analysis of Symbiotic Neurons

In section 2, we hypothesized that SANE's neurons will not converge to a single type, but will instead form several subpopulations that each fill an important role in a neural network. The experiments described in this section illustrate this process and, in the context of the Khepera simulator, describe how and why certain neuron specializations emerge.

5.1 Principal Component Analysis

In order to visualize SANE's populations, a method for displaying the neurons in 2-D is needed. Principal component analysis (PCA) (see e.g., Jolliffe, 1986) is a useful tool for visualizing the relative distances between high-dimensional vectors. PCA performs a coordinate transformation on a set of data points. The first dimension is chosen along the direction with the most variance in the data. The second is chosen perpendicular to the first, accounting for as much remaining variance as possible. Each new dimension or *principal component* is chosen in a similar fashion. The final result is a new coordinate system that is ordered according to variance. PCA can be used to perform a *dimensionality reduction* on high-dimensional data. To reduce the data points to M dimensions, PCA is run to determine the dimensions of maximum variance. The data points are then plotted along the first M coordinates. Since the coordinates are ordered, the resulting plot accounts for as much of the data variance as possible in M dimensions.

PCA can be used to reduce the high-dimensional genetic chromosomes into two or three dimensions, which can then be plotted. There are several problems, however, with this approach. First, in PCA reductions on several 240-dimensional vector populations, the first two principal components were able to capture only 60% of the data variance. More components are needed to accurately represent this data, but such representations cannot be plotted. Second, a PCA plot of the raw genetic chromosomes may not accurately represent functional similarities among the neurons. Similar genotypes (chromosomes) may not always produce similar phenotypes (neural network hidden units). For example, two neurons may have identical bits in their weight alleles, but a few differing bit positions in their label alleles can create vastly different network architectures. A PCA plot based on the chromosomes of these two neurons would place them close to each other, when they actually function quite differently.

A more appropriate strategy is to compute a *function vector* for each neuron that describes its role in a neural network. The function vector can then be reduced using a PCA and plotted, resulting in a more accurate visualization of the different neuron roles. To generate such a vector, a neuron is implemented as the only hidden unit in a network and the network's output layer activations are recorded (table 4). The function vector captures the neuron's responses to each input unit activation. Thus, unlike the basic chromosome vector, the function vector represents the direct behavior of each neuron in a neural network.

The function vector can be post-processed further to produce a functional representation in terms of the task. For example, in the Khepera task the output layer activations can be interpreted as motor commands. Using the output layer definition of section 4.1, activations $[0.4, 1.0, 0.6, 0.2]$ translate into motor commands of $[2, -8]$[5]. Post-processing the function vector in this way creates a more accurate representation of the effect each neuron has on the robot. The final function vector

[5]For each motor (left and right) take the difference in the positive and negative direction and multiply by the maximum motor activation: $(0.6 - 0.4) \times 10 = 2$; $(0.2 - 1.0) \times 10 = -8$.

1. Initialize the function vector to nil.
2. Build a neural network with the neuron as the only hidden unit.
3. For each input unit i:
   a. Set input i to 1.0 and all others to 0.0
   b. Propagate the activation through the output layer
   c. Append the output layer activations to the function vector

Table 4: The steps to compute a functional representation of a neuron. The function vector represents the actual function the neuron performs within the neural network.

consists of only 18 dimensions which can be reduced to 2 dimensions through PCA while preserving 95% of the variance.

Figures 9 plots the two-dimensional functional representations of the neuron populations from a simulation in the Khepera task. Other simulations produced similar plots (Moriarty, 1997). Snapshots of the populations were taken at generations 0, 10, 20, 40, and 80. Generation 0 shows a fairly uniform distribution reflective of the initially random populations. As the populations evolve, neurons begin to cluster together and form subpopulations or specializations. In the final generation, the specializations are very distinct. The last graph plots the neurons that were included in the top three networks of the last generation. The graph shows that the best networks utilized neurons from several different subpopulations. Such diversity demonstrates that each of the specializations plays an active role in the best neural networks.

In addition to the subpopulation clusters, each plot contains several examples of neurons that are located between clusters and isolated from other neurons. Such neurons are created by inter-breeding two members of different specializations and thus contain some of the functionality of each. The isolated neurons in the PCA demonstrates how SANE can build new neuron roles through inter-breeding existing roles. In other words, these neurons are the pioneers that explore new neuron roles. If an effective role is found, a new subpopulation of neurons will form around the isolated neuron. An example of this phenomenon can be seen in figure 9. In generation 10, a single neuron exists around the point (27,2). As the population evolves, more and more neurons begin to cluster around this area, which leads to the conclusion that the original neuron discovered a valuable neuron role.

The PCA analysis confirms the hypothesis that SANE evolves several different types of neurons in a single population. As the populations evolve, neurons merge into several different specializations that optimize different aspects of the neural networks. Moreover, the specializations provide sufficient diversity to form new neural structures by inter-breeding between specializations. This phenomena is quite unique in evolutionary algorithms, since most approaches to coevolution evolve separate species in segregated subpopulations or islands. In contrast, SANE forms its specializations naturally in a single population and takes advantage of the inherent diversity to produce a more explorative search.

5.2 Lesion Studies

While the emergence of specializations is clear from the PCA studies, the function of each and the overall division of labor is not. To better understand the role of each specialization in the neural networks, simulated lesion studies were conducted. Lesions are used in biological neural networks

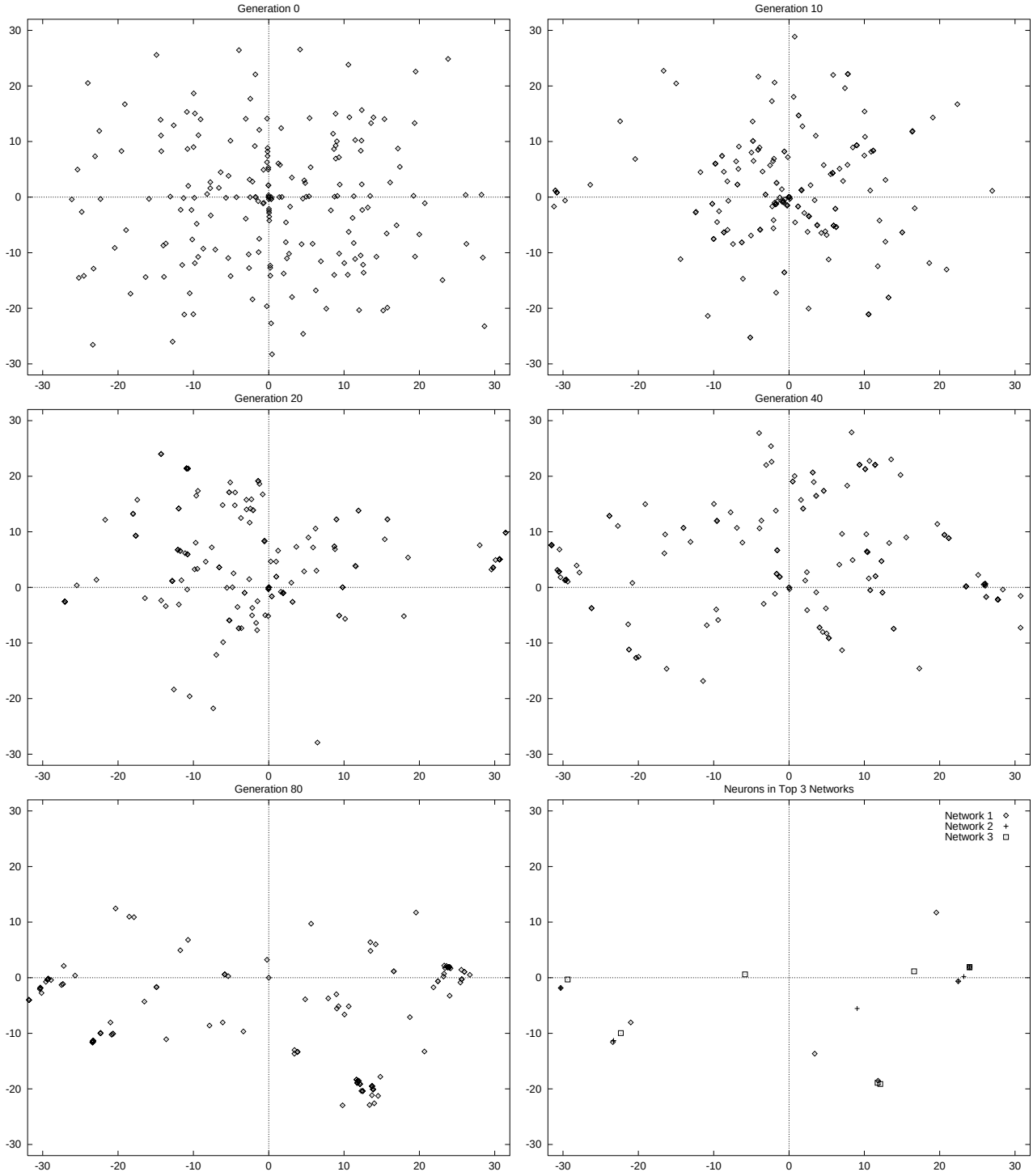

Figure 9: PCA of simulation 1. The position on one PCA graph does not necessarily correspond to the same position on another PCA graph because dimensions are determined separately in each plot. However in these experiments, this has in general held true. Thus, the neurons around point 25,2 in generation 10 are similar in function to the neurons around point 25,2 in generation 80.

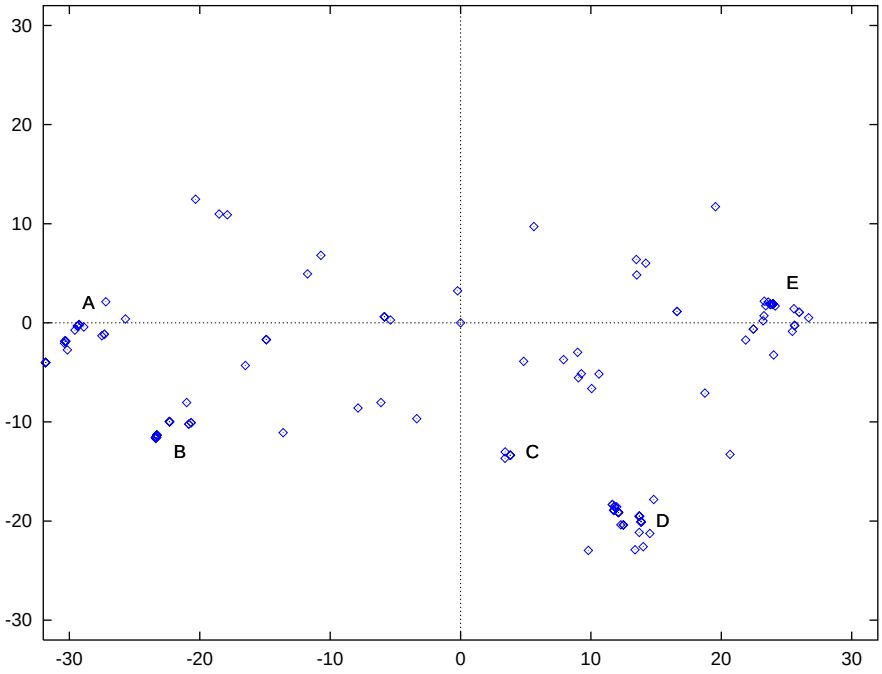

Figure 10: The principle components analysis (PCA) of the neuron population in the final generation. The specializations that are included in the top network are labeled A through E.

to identify functions or functional areas of the brain. If a lesion results in a loss of a brain function, it can be concluded that the lesioned area was involved in carrying out that function. Similar experiments can be performed in artificial neural networks, by removing neurons and observing the behavior of the modified network.

Figure 10 shows a close up of the final PCA of the population for the first simulation. The specializations that are represented in the best network of the final generation are labeled A through E. While other groupings of neurons are also present, the current lesion experiments are only interested in the specializations included in the top neural network.

Two types of lesion experiments were conducted for a specific neuron or specialization: a neuron capability test and a neuron necessity test. In the capability test, a neuron is implemented alone and thus forms the entire hidden layer. Such experiments are designed to show how functional each neuron is without aid from other neurons in the population. The capability test will effectively demonstrate that a single neuron can not perform the entire task on its own. The second test is a necessity test, where a neuron in a functioning neural network is removed. The necessity test shows how crucial a specific neuron is in the performance of a neural network.

Table 5 shows the neurons included in the final network (numbered by fitness rank), their corresponding specializations, and the performance in the two lesion tests. The network contains two neurons from specialization B and two from E. Neuron number 1, which is represented by the single point at (19,11) in figure 10, does not appear to be a member of a well-defined specialization and is therefore not given a specialization label.

As expected, no neuron formed an effective hidden layer by itself. In the capability test, the highest scoring neuron only achieved a performance level of 64.5 (average Euclidean distance in the Khepera task), which is 1/6 of the performance of the entire network of neurons. The neurons

Neuron Rank	Specialization	Capability	Necessity
48	C	64.5	382.9
16	A	6.0	305.7
11	E	11.8	402.6
46	B	12.3	341.6
1	-	10.8	294.6
38	E	12.3	326.0
40	D	44.2	381.4
39	B	13.3	293.0

Table 5: Lesion results from simulation 1. Each of the neurons in the top network are implemented alone (Capability) and lesioned from the network (Necessity). The resulting network is tested in the Khepera task. The complete (non-lesioned) network achieved a performance level of 409.5. The capability test shows that no neuron can perform the entire task alone. The necessity test shows that while the network is quite robust to damage, each neuron plays an important role.

Lesioned Specialization	Performance
A	305.7
B	207.0
C	382.9
D	381.4
E	145.3

Table 6: Specialization Lesions. All members of a specialization are removed from the network. The resulting network is tested in the Khepera task. The complete (non-lesioned) network achieved a performance level of 409.5. Each specialization is crucial to achieve the network's top performance.

generally spun the robot rapidly in either direction. Neurons 48 and 40, however, did not spin the robot, but rather moved in a straight line. Unfortunately, these two neurons also moved straight into the nearest wall. The capability test numbers clearly illustrate the symbiotic nature of SANE's neuron population. The specializations that each represent cannot function without the presence of the other neurons in the hidden layer.

The necessity tests show the importance of each neuron to the network. Each neuron lesion caused a performance drop, in some cases as much as 25%. This consistent drop demonstrates that each neuron serves an important role. Interestingly, neurons of similar specializations did not exhibit similar performance degradations when lesioned. For example, neurons 11 and 38 are both members of specialization E. When lesioned from the network, the resulting performances are 402.6 without neuron 11 and 326.0 without neuron 38.

An explanation for this disparity is that neuron 38 encompasses neuron 11's function and that neuron 11 is a weak member of specialization E. To confirm this hypothesis, experiments were conducted where entire specializations were lesioned from the network. Figure 6 summarizes the performance numbers of the specialization lesion experiments. When both members of specialization E were lesioned, the network experienced a dramatic performance drop. The difference between the performance when both members are lesioned (145.3) and the performance when only neuron 38 is lesioned (326), confirms that neuron 11 does provide some of the function of specialization E. However, given the results of the previous necessity test, the function is not enough to compensate for the loss of neuron 38. Thus, neuron 11 does appear to be a weak member of specialization E.

Neuron Rank	Specialization	Motor Activation from Specific Sensors				
		None	Left	Forward	Right	Rear
48	C	+3,+3	0,0	+3,+3	+3,+3	+3,+3
16	A	-10,+8	-4,+2	-10,+8	-10,+8	-10,+8
11	E	+9,-5	+9,-5	+5,-3	+1,-1	+9,-5
46	B	-5,+8	-5,-6	-5,+8	-5,+8	-5,+8
1	-	+4,-7	+4,-9	+4,-9	+1,-4	+4,-8
38	E	+9,-5	+9,-5	+9,-5	0,-1	+9,-5
40	D	+9,+4	+9,+4	0,0	+5,+2	+9,+4
39	B	-5,+8	-3,+3	-5,+8	-5,+9	-5,+9

Table 7: Individual neuron responses to specific sensory inputs. The output numbers refer to the speed at which the neuron drives the left and right motors. For example, when neuron 11 receives input from its front sensors, it generates motor outputs of +5 and -3.

To better understand the actual function of each neuron, the motor outputs, in response to specific sensory input, were analyzed. Table 7 shows the motor responses of each neuron given sensor activations on the left, in front, on the right, and behind the robot. The motor outputs given no activation are also shown. The two output numbers represent the activation of each of the robot's motors.

Each neuron appears to give attention to a single sensor direction and change its output only when that particular sensor becomes active. For example, neuron 48 produces a consistent motor activation of +3 for both the left and right motors, except when the left sensors are active. Such output changes are very illustrative of the neuron's function, because they demonstrate the sensors to which the neuron responds and the type of response that is elicited.

As described in the capability test, neurons 40 and 48 cause the robot to move in a straight line when implemented alone. This behavior is represented in their motor output activations in table 7. Both neurons provide consistent positive activations to both motors, which causes the robot to move forward. The difference between the two neurons, which constitutes the difference between specialization C and D, is that they are sensitive to different sensory input. Neuron 40 reduces its positive motor activations when a left sensor is activated and neuron 48 does so when a forward sensor is activated. The elicited response of both neurons is to slow the robot when the sensor is activated. These two neurons provide the thrust of the robot on long straightaways and then slow the robot when obstacles are present. Neither of the neurons provide actual obstacle avoidance behavior; they are merely responsible for slowing the robot. Another specialization performs the turning.

Neurons 11 and 38, members of specialization E, and neuron 1 provide strong positive impulses to the left motor and strong negative impulses to the right motor. Naturally, these activations cause the robot to spin very fast in a clockwise direction. Specialization E, however, has evolved a symbiotic relationship with specializations A, B, and D, which spin the robot in a counterclockwise direction. The net result is a robot that does not spin. The balance that is attained between all of the motor activations explains the performance drop off when any one neuron is excluded.

Specialization E responds to activations of the front and right sensors by reducing the left motor and increasing the right motor. Such activations reduce the clockwise spin impulse and the robot begins to turn in a counterclockwise direction. The effect of specialization E is thus to veer the robot to the left to avoid objects in front of and to the right. Specialization A is a mirror of

specialization E, veering the robot to the right to avoid objects on the left.

Neuron 1 was not placed in a specific specialization, because in the PCA graph it was not plotted in a distinct subgroup. Its proximity to specialization E on the PCA, however, is exhibited in its motor responses. Like specialization E, it reduces its normal clockwise spin impulse when the right sensors are active. The normal spin impulse and modified spin impulse of neuron 1, however, are not nearly as strong as specialization E. Neuron 1 was likely a mutant of specialization E that has increased the spin impulses just enough to make the robot make right turns in response to objects on the left. As shown in the necessity test, the network performs poorly when neuron 1 is removed.

Specialization B has a similar relationship to specialization A as neuron 1 does to specialization E. Like specialization A, B neurons spin the robot in a counter clockwise direction and reduce the spin impulse when a left sensor is active. The magnitude of the spin reduction, however, is much less than specialization A and appears to be more of a fine tuning of the right turn.

In summary, the roles of the neurons in the top network are:

48	Provides forward activations and slows down when object is sensed on the left.
16	Veers the robot right to avoid objects to the left.
11	Veers the robot left to avoid objects to the right.
46	Provides small right impulse when objects are sensed to the left.
1	Provides additional spin impulses for specialization E.
38	Veers the robot left to avoid objects to the right.
40	Provides forward activations and balances the spin caused by the other neurons. Slows robot when an object is sensed in front and to right.
39	Veers the robot right to avoid objects to the left.

Lesion studies conducted in other Khepera simulations produced similar results (Moriarty, 1997). In almost all cases, separate specializations evolved to control the forward thrust, right turns, and left turns. At least one simulation evolved a neuron specialization that produced a 0,0 motor activation across all sensory inputs. These "no-op" neurons have no effect on the network behavior and may have evolved as fillers, since the existing specializations were sufficient for solving the task.

It is also important to note that neuron roles were often redundant across several specializations and within the neural networks. For example, specializations A and B both maneuver around objects to the left of the robot, and members of both are included in the final network. This suggests that SANE does not develop a minimal set of behaviors, but instead distributes important functions across several neurons and multiple specializations. This produces a much more robust controller, since damage to a single neuron can be overcome by the redundant function of the other neurons. As shown in table 5, the loss of one neuron does not result in a dramatic drop in performance.

The lesion studies demonstrate the symbiotic nature of SANE's neurons. No neuron could perform well alone, and every neuron was necessary to achieve the network's top performance level. The studies also confirm the hypothesis that SANE performs a parallel search in several different subgroups of its neuron population. Analysis of the roles of the different specializations show a

clear division of labor with the largest division between forward motor neurons, left turn neurons, and right turn neurons. Several subpopulations within specializations appear to be refinements of the major roles, which allow SANE to continue to improve each specialization. It is this parallel search that sets SANE apart from existing neuro-evolution approaches and should allow SANE to more efficiently solve difficult problems.

6 Related Work

This section relates SANE's symbiotic evolution to other cooperative, coevolutionary approaches. While there are numerous examples of coevolution in the literature, we highlight those methods that maintain diversity and/or decompose the problem by evolving partial solutions that cooperate to form a global solution.

The search behavior of symbiotic evolution is similar to the implicit fitness sharing in the coadaptive genetic algorithms of Smith et al. (1993) and Smith and Gray (1993). In their immune system model, Smith and Gray (1993) evolved artificial antibodies to recognize or match artificial antigens. Since each antibody can only match one antigen, a diverse population of antibodies is necessary to guard against a variety of antigens.

The coadaptive genetic algorithm model is based more on competition than cooperation. Each antibody must compete for survival with other antibodies in the subpopulation to recognize the given antigen. The fitness of each individual reflects how well it matches its opposing antigen, not how well it cooperates with other individuals. The antibodies are thus not dependent on other antibodies for recognition of an antigen and only interact implicitly through competition. Horn et al. (1994) characterize this difference as weak cooperation (coadaptive GA) vs. strong cooperation (symbiotic evolution). Since both approaches appear to have similar effects in terms of population diversity and speciation, further research is necessary to discover the relative strengths and weaknesses of each method.

Smith and Cribbs (1994) have proposed a method where a learning classifier system (LCS) can be mapped to a neural network. Each hidden node represents a classifier rule that must compete with other hidden nodes in a winner-take-all competition. Like SANE, the evolution in the LCS/NN is performed on the neuron level instead of at the network level. Unlike SANE, the LCS/NN is a pure "Michigan" approach where the entire population of neurons represents the final solution. In SANE, subpopulations represent the solution.

The LCS/NN implementation uses a variant of the cascade correlation algorithm (Fahlman & Lebiere, 1990) to compute fitness levels for each neuron. Neuron fitness levels are increased if their activations correlate with correct output from the neural network. However, by basing credit assignment on the known correct behavior, the current LCS/NN implementation cannot be used for reinforcement learning, which is the primary direction of SANE. In most sequential decision tasks, correct behavior is unknown.

Potter and De Jong have developed a symbiotic evolutionary strategy called Cooperative Coevolutionary Genetic Algorithms (CCGA) and have applied it to both neural network and rule-based systems (Potter, 1997; Potter & De Jong, 1995; Potter et al., 1995). The CCGA evolves partial solutions much like SANE, but distributes the individuals differently. Whereas SANE keeps all individuals in a single population, the CCGA evolves specializations in distinct subpopulations or *islands*. Members of different subpopulations do not inter-breed across subpopulations, which eliminates haphazard, destructive recombination between dominant specializations, but also removes

information-sharing between specializations.

Evolving in distinct subpopulations places a heavier burden on *a priori* knowledge of the number of specializations necessary to form an effective complete solution. In SANE, the number and distribution of the specializations is determined implicitly throughout evolution. For example, a network may be given eight hidden neurons but may only require four *types* of hidden neurons. SANE would evolve four different specializations and redundantly select two from each for the final network. While two subpopulations in the CCGA could represent the same specialization, they cannot share information and therefore are forced to find the redundant specialization independently. Potter and De Jong (1995) have proposed a method that automatically determines the number of partial solutions necessary by incrementally adding random subpopulations. This approach appears promising, and motivates further research comparing the single population and incremental subpopulation approaches.

7 Applications of SANE

In practice, the evolutionary search in SANE appears to scale well to real world applications. This section describes three such applications that demonstrate SANE's scope and its scale up potential. Each of these applications have used the same approach described in this paper. The only domain-specific engineering occurred in the input and output layer representations.

Game Tree Search (Moriarty & Miikkulainen, 1994). Minimax search is currently the standard method for game tree searching. Unfortunately, minimax relies on heuristic evaluation functions that are often inaccurate and misleading. They generate errors that propagate through the tree and can cause minimax to select poor moves. SANE was implemented to generate neural networks that serve as filters for minimax, allowing it to see only information that lead to good decisions. The SANE networks were implemented in the former world champion Othello program Bill (Lee & Mahajan, 1990) and significantly improved its performance. SANE's application to game playing demonstrates how it may be used to develop new advances in widely studied problems.

Controlling Chaos (Weeks & Burgess, 1997). Weeks and Burgess applied SANE to the difficult task of controlling chaos in unstable systems. Chaos is dynamical behavior that is unpredictable over long periods of time, but obeys simple laws. Chaos can be controlled by applying small perturbations to system variables to achieve stability around a fixed point. Once the chaotic behavior is controlled, future system behavior can be more easily and accurately predicted. Weeks and Burgess demonstrated how SANE can efficiently evolve neural networks to control chaos in several unstable systems. Unlike existing methods that require knowledge of the underlying system dynamics, SANE's formed networks simply through trial and error experimentation using only the relative stability of the system as feedback. Moreover, SANE is the only method that has been shown to stabilize a system that is far from its stable state. The results show that SANE is quite effective at controlling chaotic behavior and should be applicable to many unstable systems including systems that are poorly understood.

Robot Arm Control (Moriarty & Miikkulainen, 1996b). Most neural network applications to robot arm control learn hand-eye coordination through supervised training methods such as backpropagation or conjugate gradient descent. Supervised learning, however, requires training examples that demonstrate correct mappings from input to output. The current approaches for generating training examples for robot arm control are limited and ineffective in uncertain or obstacle-filled domains. Thus, neuro-controllers that learn from supervised techniques cannot in-

tegrate target reaching with obstacle avoidance. SANE, however, does not require input/output examples and can learn the intermediate joint rotations necessary for avoiding obstacles simply by trial and error movements. Applied to a simulation of the OSCAR-6 robot arm, SANE learned to maneuver around random obstacles to reach randomly place targets. To our knowledge, SANE is the only learning system that has successfully combined both target reaching and obstacle avoidance strategies for a robot arm.

8 Future Work

Symbiotic evolution is not unique to connectionist systems, but may provide useful insight in other areas of machine learning as well. One application is to employ symbiotic evolution to evolve a rule base for multi-category classification. Current machine learning techniques do not directly induce shared intermediate concepts between multiple categories, but instead typically re-invent intermediate states for each category. For example, in an animal classification domain, the mammal concept is normally not shared between zebra and giraffe, but is learned separately for each specific mammal. Shared concepts, however, are advantageous because they can increase the classification accuracy for each category by applying general knowledge attained about one category to a related, but possibly more unfamiliar category (Ourston & Mooney, 1994).

Symbiotic evolution, however, is capable of forming shared intermediate concepts by simultaneously evolving rules which are used to classify multiple categories. From an initially random rule base, subpopulations of rules could be selected to form a domain theory. The domain theory could then be evaluated through theory refinement (Ourston & Mooney, 1994) which measures both the accuracy of the domain theory and the amount of refinement necessary. The evaluation score of the domain theory would be given to each participating rule and the process of selecting and evaluating random subpopulations would repeat. Once each rule has an average utility measure, crossover and mutation operators would be applied to form a new rule base. Since sharing intermediate states generally requires less theory refinement and can produce more accurate classifiers, evolutionary pressures will select cooperative rules which connect together and form shared intermediate concepts.

While we believe that symbiotic evolution is a general principle, applicable not only to neural networks but to other representations as well, not all representations may be compatible with this approach. Symbiosis emerges naturally in the current representation of neural networks as collections of hidden neurons, but preliminary experiments with other types of encodings, such as populations of individual network connections, have been unsuccessful (Steetskamp, 1995). An important facet of SANE's neurons is that they form complete input to output mappings, which makes every neuron a primitive solution in its own right. SANE can thus form subsumption-type architectures (Brooks, 1991), where certain neurons provide very crude solutions and other neurons perform higher-level functions that fix problems in the crude solutions. Preliminary studies in simple classification tasks have uncovered some subsumptive behavior among SANE's neurons. An important focus for future research will be to further analyze the functions of evolved hidden neurons and to study other symbiotic-conducive representations.

9 Conclusion

Cooperative, coevolutionary algorithms offer a promising alternative to the standard EA methods in dynamic control tasks. Coevolutionary methods allow for more aggressive searches for solutions, while discouraging convergence on suboptimal solutions. Moreover, the diverse populations adapt more readily to any fluctuations in the environment. The SANE system incorporates a coevolutionary search strategy called symbiotic evolution to form neural networks in difficult decision tasks. Compared to standard neuro-evolutionary approaches in a mobile robotics task, SANE formed solutions faster, maintained higher level of population diversity, and was more adaptive in domain shifts. Principal component analysis and lesion studies illustrate how SANE's neurons specialize within the single population and fill several different roles. This specialization is the heart of SANE's search efficiency. By reducing the solution space for each individual, SANE searches in parallel decompositions of the complete neural network space, while maintaining high levels of diversity. This coevolutionary search coupled with the outer loop blueprint evolution is quite promising and should extend well to more difficult problems.

References

Belew, R. K., McInerney, J., & Schraudolph, N. N. (1991). Evolving networks: Using genetic algorithm with connectionist learning. In Farmer, J. D., Langton, C., Rasmussen, S., & Taylor, C. (Eds.), *Artificial Life II* Reading, MA. Addison-Wesley.

Brooks, R. A. (1991). Intelligence without representation. *Artificial Intelligence*, *47*, 139–159.

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

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

Fahlman, S. E., & Lebiere, C. (1990). The cascade-correlation learning architecture. In Touretzky, D. S. (Ed.), *Advances in Neural Information Processing Systems 2*, pp. 524–532. Morgan Kaufmann, San Mateo, CA.

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

Goldberg, D. E., & Deb, K. (1991). A comparative analysis of selection schemes used in genetic algorithms. In Rawlins, G. (Ed.), *Foundations of Genetic Algorithms*, pp. 69–93. Morgan-Kaufmann.

Goldberg, D. E., & Richardson, J. (1987). Genetic algorithms with sharing for multimodal function optimization. In *Proceedings of the Second International Conference on Genetic Algorithms*, pp. 148–154 San Mateo, CA. Morgan Kaufmann.

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

Holland, J. H. (1975). *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence*. University of Michigan Press, Ann Arbor, MI.

Horn, J., Goldberg, D. E., & Deb, K. (1994). Implicit niching in a learning classifier system: Nature's way. *Evolutionary Computation*, *2*(1), 37–66.

Jolliffe, I. T. (1986). *Principal Component Analysis*. Springer-Verlag, New York, NY.

Koza, J. R., & 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, pp. 397–404 New York, NY. IEEE.

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

Michel, O. (1995). Khepera simulator version 1.0 user manual. http://wwwi3s.unice.fr/ om/khep-sim.html.

Mondada, F., Franzi, E., & Ienne, P. (1993). Mobile robot miniaturization: A tool for investigation in control algorithms. In *Proceedings of the Third International Symposium on Experimental Robotics*, pp. 501–513 Kyoto, Japan.

Moriarty, D. E. (1997). *Symbiotic Evolution of Neural Networks in Sequential Decision Tasks*. Ph.D. thesis, Department of Computer Sciences, The University of Texas at Austin.

Moriarty, D. E., & Miikkulainen, R. (1994). Evolving neural networks to focus minimax search. In *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94)*, pp. 1371–1377 Seattle, WA. MIT Press.

Moriarty, D. E., & Miikkulainen, R. (1996a). Efficient reinforcement learning through symbiotic evolution. *Machine Learning*, *22*, 11–32.

Moriarty, D. E., & Miikkulainen, R. (1996b). Evolving obstacle avoidance behavior in a robot arm. In *From Animals to Animats: Proceedings of the Fourth International Conference on Simulation of Adaptive Behavior (SAB-96)*, pp. 468–475 Cape Cod, MA.

Nolfi, S., & Parisi, D. (1992). Growing neural networks. In *Artificial Life III* Reading, MA. Addison-Wesley.

Ourston, D., & Mooney, R. J. (1994). Theory refinement combining analytical and emprical methods. *Artificial Intelligence*, *66*, 311–344.

Paredis, J. (1995). Coevolutionary computation. *Artificial Life*, *2*, 355–375.

Potter, M. A. (1997). *The Design and Analysis of a Computational Model of Cooperative Coevolution*. Ph.D. thesis, George Mason University.

Potter, M. A., & De Jong, K. A. (1995). Evolving neural networks with collaborative species. In *Proceedings of the 1995 Summer Computer Simulation Conference* Ottawa, Canada.

Potter, M. A., De Jong, K. A., & Grefenstette, J. (1995). A coevolutionary approach to learning sequential decision rules. In Eshelman, L. (Ed.), *Proceedings of the Sixth International Conference on Genetic Algorithms* Pittsburgh, PA.

Smith, R. E., & Cribbs, H. B. (1994). Is a learning classifier system a type of neural network?. *Evolutionary Computation*, *2*(1).

Smith, R. E., Forrest, S., & Perelson, A. S. (1993). Searching for diverse, cooperative populations with genetic algorithms. *Evolutionary Computation*, *1*(2), 127–149.

Smith, R. E., & Gray, B. (1993). Co-adaptive genetic algorithms: An example in othello strategy. Tech. rep. TCGA 94002, Department of Engineering Science and Mechanics, The University of Alabama.

Steetskamp, R. (1995). Explorations in symbiotic neuro-evolution search spaces. Masters Stage Report, Department of Computer Science, University of Twente, The Netherlands.

Weeks, E. R., & Burgess, J. M. (1997). Evolving artificial neural networks to control chaos. *Phys. Rev. E*, *56*, 1531–1540.

Whitehead, B. A., & Choate, T. D. (1995). Cooperative competitive genetic evolution of radial basis function centers and widths for time series prediction. *IEEE Transactions on Neural Networks*.

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