Cooperative Coevolution: An Architecture for Evolving Coadapted Subcomponents

Mitchell A. Potter
Navy Center for Applied Research
in Artificial Intelligence
Naval Research Laboratory
Washington, DC 20375, USA
mpotter@aic.nrl.navy.mil

Kenneth A. De Jong
Computer Science Department
George Mason University
Fairfax, VA 22030, USA
kdejong@cs.gmu.edu

Abstract

To successfully apply evolutionary algorithms to the solution of increasingly complex problems, we must develop effective techniques for evolving solutions in the form of interacting coadapted subcomponents. One of the major difficulties is finding computational extensions to our current evolutionary paradigms that will enable such subcomponents to "emerge" rather than being hand designed. In this paper, we describe an architecture for evolving such subcomponents as a collection of cooperating species. Given a simple string-matching task, we show that evolutionary pressure to increase the overall fitness of the ecosystem can provide the needed stimulus for the emergence of an appropriate number of interdependent subcomponents that cover multiple niches, evolve to an appropriate level of generality, and adapt as the number and roles of their fellow subcomponents change over time. We then explore these issues within the context of a more complicated domain through a case study involving the evolution of artificial neural networks.

Keywords

Coevolution, genetic algorithms, evolution strategies, emergent decomposition, neural networks.

1 Introduction

The basic hypothesis underlying the work described in this paper is that to apply evolutionary algorithms (EAs) effectively to increasingly complex problems, explicit notions of modularity must be introduced to provide reasonable opportunities for solutions to evolve in the form of interacting coadapted subcomponents. A good example of such problems is behavior learning tasks, such as those one would encounter in the domain of robotics, where complex behavior can be decomposed into simpler subbehaviors. What makes finding solutions to problems of this sort especially difficult is that a natural decomposition often leads to interacting subcomponents that are highly dependent on one another. This makes coadaptation a critical requirement for their evolution.

There are two primary reasons traditional EAs are not entirely adequate for solving these types of problems. First, the population of individuals evolved by these algorithms has a strong tendency to converge in response to an increasing number of trials being allocated to observed regions of the solution space with above average fitness. This strong convergence property precludes the long-term preservation of diverse subcomponents, because any but the strongest individual will ultimately be eliminated. Second, individuals evolved

by traditional EAs typically represent complete solutions and are evaluated in isolation. Since interactions between population members are not modeled, there is no evolutionary pressure for coadaptation to occur. To complicate matters further, we may neither know *a priori* how many subcomponents there should be nor what roles the subcomponents should assume. The difficulty then comes in finding computational extensions to our current evolutionary paradigms in which reasonable subcomponents "emerge" rather than being designed by hand. At issue is how to identify and represent such subcomponents, provide an environment in which they can interact and coadapt, and apportion credit to them for their contributions to the problem-solving activity such that their evolution proceeds without human involvement.

This paper is organized as follows. Section 2 begins with a discussion of some important issues related to evolving coadapted subcomponents and describes some previous approaches to extending the basic evolutionary model for this purpose. This is followed in Section 3 by a description of our architecture for evolving coadapted subcomponents as a collection of cooperating species. Section 4 investigates whether reasonable subcomponents will automatically arise through the use of such systems by attempting to show that evolutionary pressure to increase the overall fitness of the ecosystem can provide the needed stimulus for the emergence of an appropriate number of interdependent subcomponents that cover multiple niches, are evolved to an appropriate level of generality, and adapt as the number and roles of their fellow subcomponents change over time. Section 5 further explores the emergence of coadapted subcomponents through a case study in neural network evolution. Section 6 summarizes the main contributions of this paper and discusses directions for future research.

2 Evolving Coadapted Subcomponents

If we are to extend the basic computational model of evolution to provide reasonable opportunities for the emergence of coadapted subcomponents, we must address the issues of problem decomposition, interdependencies between subcomponents, credit assignment, and the maintenance of diversity.

Problem decomposition consists of determining an appropriate number of subcomponents and the role each will play. For some problems, an appropriate decomposition may be known *a priori*. Consider the problem of optimizing a function of K independent variables. It may be reasonable to decompose the problem into K subtasks, with each assigned to the optimization of a single variable. However, there are many problems for which we have little or no information pertaining to the number or roles of subcomponents that, ideally, should be in the decomposition. For example, if the task is to learn classification rules for a multimodal concept from preclassified examples, we probably will not know beforehand how many rules will be required to cover the examples or which modality of the concept each rule should respond to. Given that an appropriate decomposition is not always obvious, it is extremely important that the EA addresses the decomposition task either as an explicit component of the algorithm or as an emergent property.

The second issue concerns the evolution of interdependent subcomponents. If a problem can be decomposed into subcomponents without interdependencies, clearly each can be evolved without regard to the others. Graphically, one can imagine each subcomponent evolving to achieve a higher position on its own fitness landscape, disjoint from the fitness landscapes of the other subcomponents. Unfortunately, many problems can only be decom-

posed into subcomponents exhibiting complex interdependencies. The effect of changing one of these interdependent subcomponents is sometimes described as a "deforming" or "warping" of the fitness landscapes associated with each of the other interdependent subcomponents (Kauffman and Johnsen, 1991). Given that EAs are adaptive, it would seem that they would be well suited to the dynamics of these coupled landscapes. However, natural selection will only result in coadaptation in such an environment if the interaction between subcomponents is modeled.

The third issue is the determination of the contribution each subcomponent is making to the solution as a whole. This is called the *Credit Assignment Problem*, and it can be traced back to early attempts by Arthur Samuel (1959) to apply machine learning to the game of checkers. For example, if we are given a set of rules for playing a two-player game, such as checkers, we can evaluate the fitness of the rule set as a whole by letting it play actual games against alternative rule sets or human opponents while keeping track of how often it wins. However, it is far from obvious how much credit a single rule within the rule set should receive given a win, or how much blame the rule should accept given a loss.

The fourth issue is the maintenance of diversity in the ecosystem. If one is using an EA to find a single individual representing a complete solution to a problem, diversity only needs to be preserved in the population long enough to perform a reasonable exploration of the search space. Typically, once a good solution is found, the EA terminates, and all but the best individual is discarded. Contrast this to evolving a solution consisting of coadapted subcomponents. In the coevolutionary paradigm, although some subcomponents may be stronger than others with respect to their contribution to the solution as a whole, all subcomponents are required to be present in the final solution.

2.1 Previous Approaches

One of the earliest extensions to the basic evolutionary model for the support of coadapted subcomponents is the *classifier system* (Holland and Reitman, 1978; Holland, 1986). A classifier system is a rule-based system in which a population of stimulus-response rules is evolved using a genetic algorithm (GA). The individual rules in the population work together to form a complete solution to a target problem. A micro-economy model, in which rules place bids for the privilege of activation, is used to handle the interactions between population members, enabling them to be highly interdependent. Credit assignment is accomplished using an algorithm called the *bucket brigade*, which passes value back to the rules along the activation chain when the system is doing well. The complex dynamics of this micro-economy model results in emergent problem decomposition and the preservation of diversity.

A more task specific approach to evolving a population of coadapted rules was taken by Giordana et al. (1994) in their REGAL system. REGAL learns classification rules consisting of conjunctive descriptions in first order logic from preclassified training examples. Problem decomposition is handled by a selection operator called *universal suffrage*, which clusters individuals based on their coverage of a randomly chosen subset $\mathcal{E}$ of the positive examples. A complete solution is formed by selecting the best rule from each cluster, which provides the necessary interaction among subcomponents. A seeding operator maintains diversity in the ecosystem by creating appropriate individuals when none with the necessary properties to cover a particular element of $\mathcal{E}$ exist in the current population. The fitness of the individuals within each cluster is a function of their consistency with respect to the set of negative examples and their simplicity. This fitness evaluation procedure effectively solves

the credit assignment problem but is highly specific to the task of concept learning from preclassified examples.

More recently, the performance of REGAL was improved through the use of semi-isolated population islands (Giordana and Neri, 1996). Each REGAL island consists of a population of conjunctive descriptions that evolve to classify a subset of the preclassified training examples. The universal suffrage operator is applied within each island population, and the REGAL seeding operator ensures that all examples assigned to an island are covered. Migration of individuals between islands occurs at the end of each generation. A supervisor process determines which examples are assigned to the islands and, occasionally, reassigns them so that each island will ultimately produce a single conjunctive description that will correctly classify all the training examples when disjunctively combined with a description from each of the other islands. As hypothesized by population geneticist Sewall Wright (1932), population islands enable the system to maintain more diversity and reach higher fitness peaks than possible with a single freely interbreeding population.

Multiple-species models, that is, those incorporating genetically isolated populations, have also been used to evolve coadapted subcomponents. Early work includes the application of a model of hosts and parasites to the evolution of sorting networks using a GA (Hillis, 1991). One species (the hosts) represents sorting networks, and the other species (the parasites) represents test cases in the form of sequences of numbers to be sorted. The interaction between the two species takes the form of complementary fitness functions. Specifically, a sorting network is evaluated on how well it sorts test cases, while the test cases are evaluated on how poorly they are sorted. Because the host and parasite species are genetically isolated and only interact through their fitness functions, they are full-fledged species in a biological sense. A two-species model has also been used to solve a number of game learning problems, including tic-tac-toe, nim, and go, by having the species represent competing players (Rosin and Belew, 1995). These competitive-species models have demonstrated that this form of interaction helps to preserve genetic diversity and results in better final solutions when compared with non-coevolutionary approaches. In addition, the credit-assignment problem is trivially solved through the use of complementary fitness functions. A limitation of these approaches, however, is their narrow range of applicability due to the requirement that the problem be hand-decomposed into two antagonistic subcomponents.

Other researchers have explored the use of cooperative-species models. Besides our approach, which will be discussed in detail throughout the rest of this paper, the coevolution of multiple cooperative species has been applied to job-shop scheduling (Husbands and Mill, 1991), and a two-species cooperative model was applied to Goldberg's three-bit deceptive function (Paredis, 1995). The decomposition used by the job-shop scheduling system was to have each species but one evolve plans for manufacturing a different component. The single remaining species evolved an arbitrator for resolving conflicts when two or more plans required the same piece of shop equipment at the same time. The two-species system for solving Goldberg's three-bit deceptive problem assigned one species to the task of evolving an effective representation in the form of a mapping between genes and subproblems and the other species to the task of evolving subproblem solutions. These two species had a symbiotic relationship in that the second species used the representations coevolved by the first species. While both of these cooperative-species models involved a hand-decomposition of the problem, our approach has emphasized emergent problem decomposition.

The application of EAs to the construction of artificial neural networks has also motivated extensions to the basic evolutionary model for the support of coadapted subcomponents. For example, in the SANE system (Moriarty, 1997; Moriarty and Miikkulainen, 1997), each individual in the population represents a single neuron by specifying which input and output units it connects to and the weights on each of its connections. A collection of neurons selected from the population constitutes a specification for constructing a complete neural network. A genetically isolated population of network *blueprints* evolves records of neurons that work well together. The neural-evaluation phase of SANE performs many cycles of selecting neurons from the population based on a blueprint, connecting the neurons into a functional network, evaluating the network, and passing the resulting fitness back to the blueprint. The fitness of a neuron is computed as the average fitness of the five best networks it participates in, which is effectively a measure of how well the neuron collaborates with other neurons in the population to solve the target problem. The relationship between the blueprint population and the neuron population is also collaborative. Rewarding individuals based on how well they collaborate results in the long term maintenance of population diversity and a form of emergent decomposition.

A more macroscopic approach has been taken by de Garis (1990) to evolve artificial neural networks for controlling simulated creatures. Rather than evolving a single large neural network to control a creature, de Garis first hand-decomposes the problem into a set of component behaviors and control inputs. A GA is then used to evolve small specialized neural networks that exhibit the appropriate subbehaviors. This is similar to the *chaining* technique pioneered by Skinner (1938) and used by the Reinforcement Learning community to train robots (Singh, 1992; Lin, 1993). Clearly, the human is very much in the loop when taking this approach.

Another biological model that has inspired the evolution of coadapted subcomponents is the vertebrate immune system (Forrest and Perelson, 1990; Smith et al., 1993; Forrest et al., 1993). Forrest et al. introduced a technique called *emergent fitness sharing* that preserves diversity in a single population and results in emergent problem decomposition by modeling some of the interactions that occur between antibodies and antigens. Emergent fitness sharing computes the fitness of each generation of antibodies by performing many iterations of a three-step process. First, a single antigen is randomly chosen from a fixed collection; second, a set of antibodies is randomly chosen from the evolving population; and third, a tournament is held to determine which of the selected antibodies matches the antigen most closely. The winner of each tournament receives a fitness increment based on the quality of the match. This model is similar in some ways to SANE, but the relationship between individuals is competitive rather than cooperative.

Finally, there has been work on extending the basic single-population evolutionary model to allow coadapted subcomponents specific to the construction of computer programs. Koza (1993), for example, has reported on the beneficial hand-decomposition of problems into a main program and a number of subroutines. Others have taken a more emergent approach through the exploration of techniques for automatically identifying blocks of useful code, generalizing them, and adapting them for use as subroutines in future generations (Rosca and Ballard, 1994, 1996).

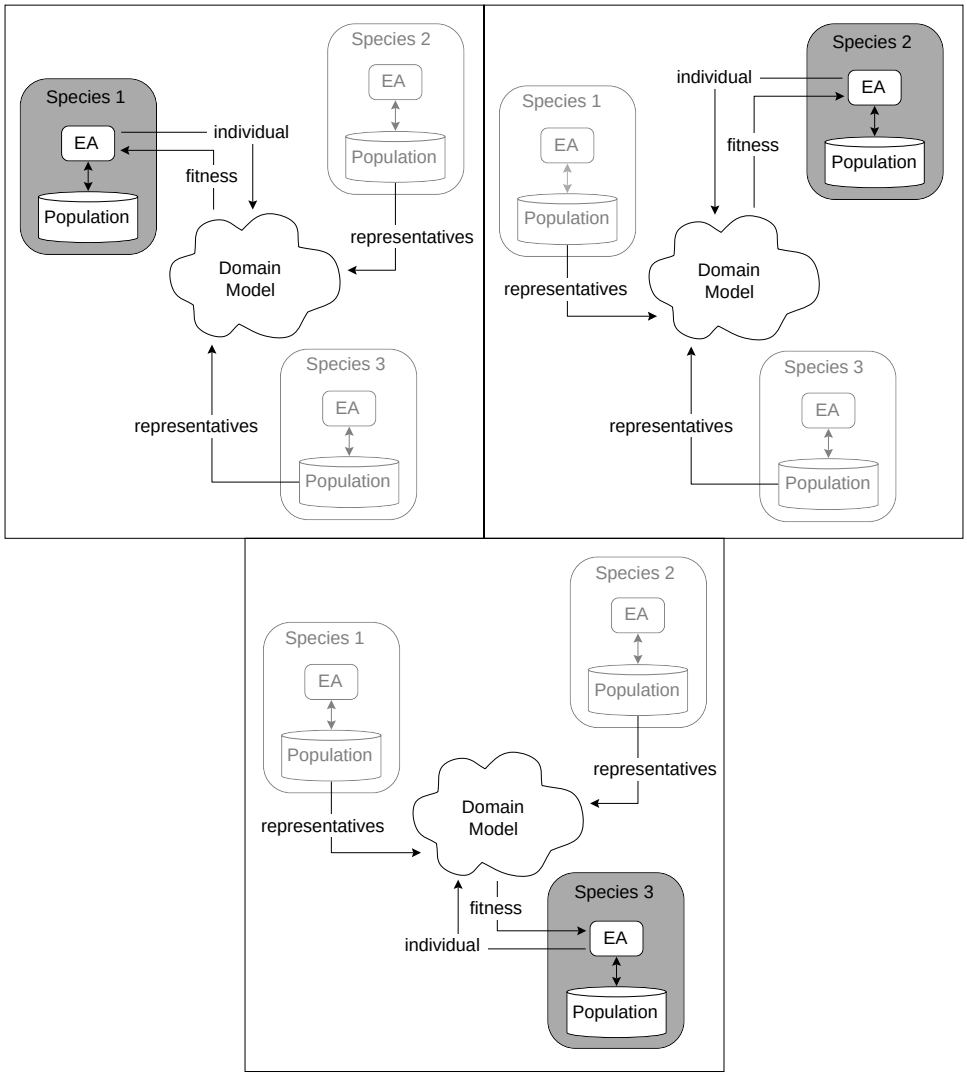

Figure 1: Coevolutionary model of three species shown from the perspective of each in turn.

3 Cooperative Coevolution Architecture

We now describe a generalized architecture for evolving interacting coadapted subcomponents. The architecture, which we call *cooperative coevolution*, models an ecosystem consisting of two or more species. As in nature, the species are genetically isolated—individuals only mate with other members of their species. Mating restrictions are enforced simply by evolving the species in separate populations. The species interact with one another within a shared domain model and have a cooperative relationship.

The basic coevolutionary model is shown in Figure 1. Although this particular illustration shows three species, the actual number in the ecosystem may be more or less. Each

species is evolved in its own population and adapts to the environment through the repeated application of an EA. Although, in principle, any EA appropriate to the task at hand can be used, we only have firsthand experience constructing coevolutionary versions of genetic algorithms and evolution strategies. The figure shows the fitness evaluation phase of the EA from the perspective of each of the three species. Although most of our implementations of this model have utilized a sequential pattern of evaluation, where the complete population of each species is evaluated in turn, the species could also be evaluated in parallel. To evaluate individuals from one species, collaborations are formed with representatives from each of the other species.

There are many possible methods for choosing representatives with which to collaborate. In some cases it is appropriate to simply let the current best individual from each species be the representative. In other cases this is too greedy and alternative strategies are preferable. For example, a sample of individuals from each species could be chosen randomly, or a more ecological approach in which representatives are chosen non-deterministically based on their fitness could be used. Alternatively, a topology could be introduced and individuals who share a neighborhood allowed to collaborate.

3.1 Examples

Some simple examples may help illustrate the architecture. One of our first uses of cooperative coevolution was to maximize a function $f(\vec{x})$ of n independent variables (Potter and De Jong, 1994). The problem was hand-decomposed into n species and each assigned to one of the independent variables. Each species consisted of a population of alternative values for its assigned variable. To evaluate an individual from one of the species, we first selected the current best individual from every one of the other species and combined them, along with the individual being evaluated, into a vector of variable values. This vector was then applied to the target function. An individual was rewarded based on how well it maximized the function within the context of the variable values selected from the other species.

As a second example, cooperative coevolution was used to develop a rule-based control system for a simulated autonomous robot (Potter et al., 1995). There were two species, each consisting of a population of rule sets for a class of behaviors. We wanted one species to evolve behaviors appropriate for beating a hand-coded robot to food pellets randomly placed in the environment and the other to evolve behaviors appropriate for the time spent waiting for food to appear. The species were seeded with some initial knowledge in their assigned area of expertise to encourage evolution to proceed along the desired trajectory. To evaluate an individual from one of the species, we selected the current best rule set from the other species, merged this rule set with the one being evaluated, and used the resulting superset to control the robot. An individual was rewarded based on how well its rule set complemented the rule set selected from the other species.

3.2 Problem Decomposition

In the previous two examples, it was known how many species were required for the task at hand and what role each should play. We took advantage of this knowledge to produce human-engineered decompositions of the problems. In other situations, it is quite possible that we may have little or no prior knowledge to help us make this determination. Ideally, we would like both the number of species in the ecosystem and the roles the species assume

to be an emergent property of cooperative coevolution.

One possible algorithm for achieving this is based on the premise that if evolution stagnates, it may be that there are too few species in the ecosystem from which to construct a good solution. Therefore, when stagnation is detected, a new species is added to the ecosystem. We initialize the species randomly and evaluate its individuals based only on the overall fitness of the ecosystem. Since we do not bias these new individuals, the initial evolutionary cycles of the species will be spent searching for an appropriate niche in which it can make a useful contribution—that is, the role of each new species will be emergent. Once a species finds a niche where it can make a contribution, it will tend to exploit this area. The better adapted a species becomes, the less likely it will be that some other species will evolve individuals that perform the same function because they will receive no reward for doing so. Conversely, if a species is unproductive, determined by the contribution its individuals make to the collaborations they participate in, the species will be destroyed. Stagnation can be detected by monitoring the quality of the collaborations through the application of the inequality

$$f(t) - f(t - L) < G \tag{1}$$

where $f(t)$ is the fitness of the best collaboration at time t, G is a constant specifying the fitness gain considered to be a significant improvement, and L is a constant specifying the length of an evolutionary window in which significant improvement must be made.

3.3 Other Characteristics of the Architecture

Evolving genetically isolated species in separate populations is a simple solution to the problem of maintaining sufficient diversity to support coadapted subcomponents. While there is evolutionary pressure for a species population to converge once it finds a useful niche, no such pressure exists for the various species to converge to the *same* niche. Rather, rewarding individuals based on how well they collaborate with representatives from the other species provides evolutionary pressure for them to make a unique contribution to the problem-solving effort. This ensures that the ecosystem will consist of a diverse collection of species.

Evolving species in separate populations also eliminates destructive cross-species mating. When individuals become highly specialized, mixing their genetic material through cross-species mating will usually produce non-viable offspring. As is demonstrated in nature, the genetic distance between two species is highly correlated with mating discrimination and the likelihood that if interspecies mating does occur the offspring will either not survive or be sterile (Smith, 1989).

Of course, the existence of separate breeding populations does not preclude interaction between species. The architecture handles interdependencies between subcomponents by evolving the species in parallel and evaluating them within the context of each other. This requires that individuals from different species be merged within a shared domain model to form a composite solution to the target problem.

Credit assignment occurs at two levels of abstraction. When evaluating the individuals within a species, the representatives from the other species remain fixed. Therefore, the fitness differential that is used in making reproduction decisions is strictly a function of the individual's relative contribution to the problem-solving effort within the context of the other species. The fitness is assigned only to the individual being evaluated, not shared with

the representatives from the other species. This greatly simplifies the credit assignment problem because there is no need to determine which species contributed what. On the other hand, we do need to occasionally estimate the level of contribution a species makes to determine whether it should be allowed to survive. This can often be accomplished by computing the fitness differential of a solution with and without the participation of the species in question.

Each species is evolved by its own EA. Communication between species is limited to an occasional broadcast of representatives, and the only global control is that required to create new species and eliminate unproductive ones. This makes parallel implementations, in which species are assigned to separate processors, trivial.

Finally, heterogeneous representations are supported. This will become increasingly important as we apply evolutionary computation to larger problems. For example, in developing a control system for an autonomous robot, some components may best be implemented as artificial neural networks, others as collections of symbolic rules, and still others as parameter vectors. The cooperative coevolution architecture enables each of these components to be represented appropriately and evolved with a suitable class of EA.

4 Analysis of Decomposition Capability

In the following empirical analysis we explore whether the algorithm outlined in Section 3.2 is capable of producing good problem decompositions purely as a result of evolutionary pressure to increase the overall fitness of the ecosystem. We will describe four studies—each designed to answer one of the following questions concerning the ability of this algorithm to decompose problems:

- Will species locate and cover multiple environmental niches?
- Will species evolve to an appropriate level of generality?
- Will adaptation occur as the number and role of species change?
- Will an appropriate number of species emerge?

This is, of course, an inductive argument. Although positive answers to these questions may not guarantee that good problem decompositions will emerge, we would certainly expect positive answers if this capability does, in fact, exist.

4.1 Coevolving String Covers

In these studies we use binary string covering as a target problem, in part because the task provides a relatively simple environment in which the emergent decomposition properties of cooperative coevolution can be explored, and because it has been used by others in related studies (Forrest et al., 1993). The problem consists of finding a set of N binary vectors that match as strongly as possible another set of K binary vectors, where K is typically much larger than N. We call these sets the *match set* and *target set* respectively. Given that K is larger than N, the match set must contain patterns shared by multiple target strings to cover the target set optimally, that is, the match set must *generalize*. The match strength S between two binary vectors $\vec{x}$ and $\vec{y}$ of length L is determined simply by summing the

number of bits in the same position with the same value as follows:

$$S(\vec{x}, \vec{y}) = \sum_{i=1}^{L} \begin{cases} 1 & \text{if } x_i = y_i \\ 0 & \text{otherwise} \end{cases} \quad (2)$$

To compute the strength of a match set M, we apply it to the target set and average the maximum computed match strengths with respect to each target set element as follows:

$$S(M) = \frac{1}{K} \sum_{i=1}^{K} \max(S(\vec{m}_1, \vec{t}_i), \ldots, S(\vec{m}_N, \vec{t}_i)) \quad (3)$$

where $\vec{m}$ and $\vec{t}$ are elements of the match set and target set respectively.

The string covering problem can be easily mapped to the model of cooperative coevolution. Recall from Figure 1 that to evaluate an individual from one species it must collaborate with representatives from each of the other species in the ecosystem. Here, individuals represent match strings, and each species will contribute one string to the match set. Representatives are chosen as the current best string from each of the other species. In other words, a match set of size N will consist of a string being evaluated from one species and the current best string from each of the other $N - 1$ species. During the initial few generations, a species may participate in collaborations before its population has ever been evaluated. In this case a random representative is chosen. The fitness of the match set will be computed from Equation 3 and assigned to the string being evaluated.

Problem decomposition, in this context, consists of determining the size of the match set and the coverage of each match set element with respect to the target strings. In the first three studies the size of the match set is predetermined. However, given that the species are randomly initialized, the coverage of each will be an emergent property of the system in all four studies.

The particular EA used in these studies is a GA. In all cases we initialize the population of each of the species randomly, use a population size of 50, two-point crossover at a rate of 0.6, bit-flipping mutation at a rate set to the reciprocal of the chromosome length, and proportionate selection based on scaled fitness.

4.2 Experimental Results

4.2.1 Locating and Covering Multiple Environmental Niches

One of the most fundamental questions concerning emergent problem decomposition is whether the method can locate and cover multiple environmental niches. In string covering, the niches we are interested in are schemata common among the target strings. By *schemata* we are referring to string templates consisting of a fixed binary part and a variable part designated by the symbol "`#`". In a previous study by Forrest et al. (1993), an experiment was performed demonstrating the ability of a traditional single-population GA to detect common schemata in a large collection of target strings. To compute the match strength between two strings, they used a linear function similar to Equation 2. Their schema detection experiment is duplicated here, with cooperative coevolution substituted for their traditional single-population GA.

The experiment consists of evolving match sets for three separate target sets, each consisting of 200 64-bit strings. The strings in the first target set will be generated in equal proportion from the following two half-length schemata:

```
11111111111111111111111111111111################################
################################11111111111111111111111111111111
```

The variable half of each of the strings will consist of random patterns of ones and zeros. Similarly, the strings in the second target set will be generated in equal proportion from the following quarter-length schemata:

```
1111111111111111########################################################
################1111111111111111########################################
################################1111111111111111########################
########################################################1111111111111111
```

and the strings in the third target set will be generated in equal proportion from the following eighth-length schemata:

```
11111111############################################################
########11111111############################################################
################11111111############################################################
########################11111111############################################################
################################11111111############################################################
########################################11111111############################################################
########################################################11111111########
############################################################11111111
```

The niches in the target set generated from the eighth-length schemata should be significantly harder to find than those generated from the half-length or quarter-length schemata because the fixed regions that define the eighth-length niches are smaller with respect to the variable region of the strings.

We know *a priori* how many niches exist and are only interested in whether we can locate and cover them, so we simply evolve an equal number of species as niches. For example, since we know that the first target set was generated from two schemata and, therefore, will contain two niches, we evolve two species to cover these niches. Similarly, four species are evolved to cover the second target set, and eight species are evolved to cover the third target set.

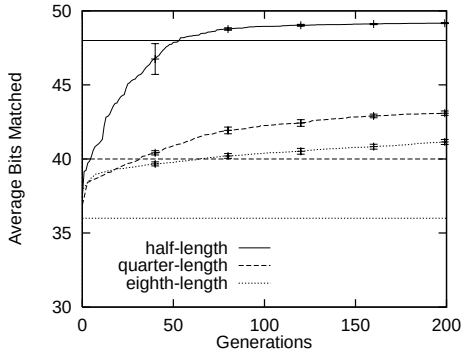

Figure 2: Finding half-length, quarter-length, and eighth-length schemata.

The average number of bits matched per target string using a match set consisting of the best individual from each species is shown in Figure 2. Each curve in the figure

was computed from the average of five runs of 200 generations using the indicated target set. Overlaid on the curves at increments of 40 generations are 95-percent confidence intervals. The horizontal lines in the graph represent the expected match values produced from the best possible single-string generalist. Given the half-length, quarter-length, and eight-length schemata shown, this generalist will consist entirely of ones, and its average match scores for the three target sets will be 48, 40, and 36 respectively. The Forrest study demonstrated that a traditional single-population GA consistently evolves this best possible single-string generalist. Our study shows that when multiple species collaborate, they are able to cover the target set better than any single individual evolved with a traditional GA. Furthermore, when more species are employed, as in the eighth-length schema experiment, the amount of improvement over the traditional GA increases.

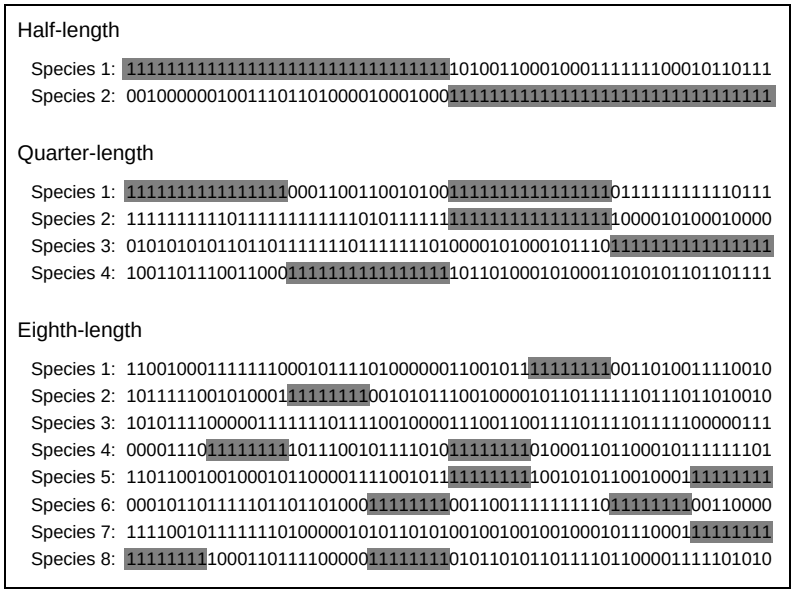

Figure 3: Final species representatives from schemata experiments.

The reason for this improvement can be seen in Figure 3. This figure shows the best individual from each of the species at the end of the final generation of the first of five runs from the half-length, quarter-length, and eighth-length schemata experiments. Substrings perfectly matching the fixed regions of the schemata are highlighted. A couple of observations can be made from this figure. First, clearly, each species focuses on one or two niches and relies on the other species to cover the remaining ones. This enables the species to cover their respective target strings better than if they had to generalize over the entire target set. Stated another way, each species locates one or two niches where it can make a useful contribution to the collaborations that are formed. This is strong evidence that the species have a cooperative relationship with one another. Second, two or more species may occasionally occupy a common niche. See, for example, the fourth and fifth species from the eighth-length schemata experiment. Although our model of cooperative coevolution does not exclude this possibility, each species must make some unique contribution to be considered viable. Third, some of the species, for example, the third species from the eighth-length schemata experiment, make no obvious contribution. It may be that this species has found a pattern that repeatedly occurs in the random region

of some of the target strings, and its contribution is simply not readily visible to us. Another possibility is that this particular species is genuinely making no useful contribution and should be eliminated.

4.2.2 Evolving to an Appropriate Level of Generality

To determine whether species will evolve to an appropriate level of generality, we generate a target set from three 32-bit test patterns and vary the number of species from one to four. Given three patterns and three or more species, each species should be able to specialize on one pattern. However, given fewer species than patterns, each must generalize to cover the patterns as best as possible. As with the previous study, this study was inspired by experiments done on emergent fitness sharing by Forrest et al. (1993).

Our test patterns are shortened complements of the ones used by Forrest as follows:

```
11111111111111111111111111111111
11111111100000000000000000000000
00000000000000000000001111111111
```

The optimal single-string cover of these patterns is the following string:

```
11111111100000000000001111111111
```

which produces an average match score of $(20+22+22)/3 = 21.33$. The optimal two-string cover of the three patterns is a string consisting of all ones and a string whose 12-bit middle segment is all zeros. A cover composed of these two strings will produce an average match score of 25.33. For example, the following two strings:

```
11111111111111111111111111111111
10010110110000000000001111110101
```

are scored as follows: $(32 + 20 + 24)/3 = 25.33$. The makeup of the extreme left and right 10-bit segments of the second string is unimportant. The optimal three-string cover of the patterns is obviously the patterns themselves.

To build on Section 4.2.1, we hid the three 32-bit test patterns by embedding them in the following three schemata of length 64:

```
1##1###1###11111##1##1111#1##1###1#1111##111111##1#11#1#11######
1##1###1###11111##1##1000#0##0###0#0000##000000##0#00#0#00######
0##0###0###00000##0##0000#0##0###0#0000##001111##1#11#1#11######
```

A target set composed of 30 strings was then generated in equal proportion from the schemata.

Four sets of five runs were performed. In the first set we evolved a cover for the target set using a single species, which is equivalent to using a traditional single-population GA. The remaining three sets of runs include evolving two, three, and four species respectively. The plots in Figure 4 show the number of target bits matched by the best individual from each species. They were generated from the first run of each set rather than the average of the five runs so that the instability that occurs during the early generations was not masked. However, we verified that all five runs of each set produced similar results. Although the

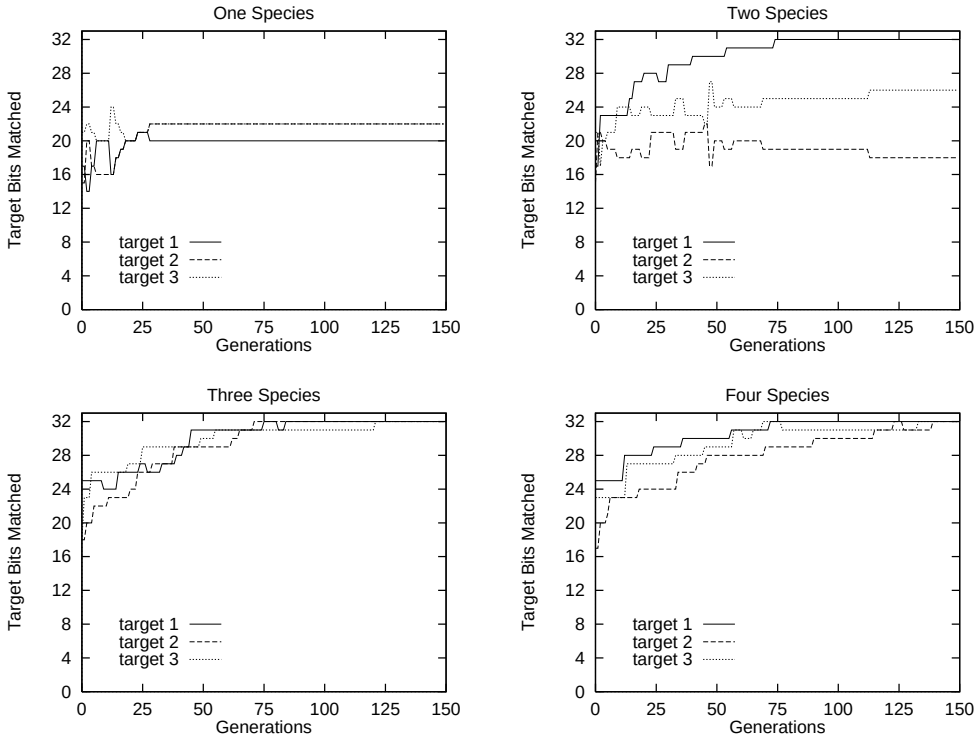

Figure 4: Covering three hidden niches with one, two, three, and four species.

figure shows just the number of bits matching the 32-bit target patterns, the fitness of individuals was based on how well all 64 bits of each target string were matched.

In the single species plot, we see that after an initial period of instability the species stabilizes at the appropriate level of generality. Specifically, 20 bits of the first pattern are matched, and 22 bits of the second and third patterns are matched. This result is consistent with the optimal single-string generalist described previously. When we increase the number of species to two, the first pattern is matched perfectly and the other two patterns are matched at the level of 26 and 18 bits respectively—consistent with the optimal two-string generalization. When three species are evolved, all three patterns are matched perfectly, indicating that each species has specialized on a different test pattern. In the four species plot, we see that when the number of species is increased beyond the number of niches, perfect matches are also achieved; however, more fitness evaluations are required to achieve this level of performance.

Figure 5 shows the best individual from each of the species at the end of the final generation. After removing all the bits corresponding to the variable regions of the target strings, the patterns that remain are the optimal one, two, and three element covers described earlier, which is conclusive evidence that the species have evolved to appropriate levels of generality. One can also see from this figure that the third and fourth species have focused on the same 32-bit target pattern. However, the bits from these two individuals corresponding to the variable regions of the target strings are quite different. What is occurring is that the two species are adapting to different repeating patterns in the variable regions of the target

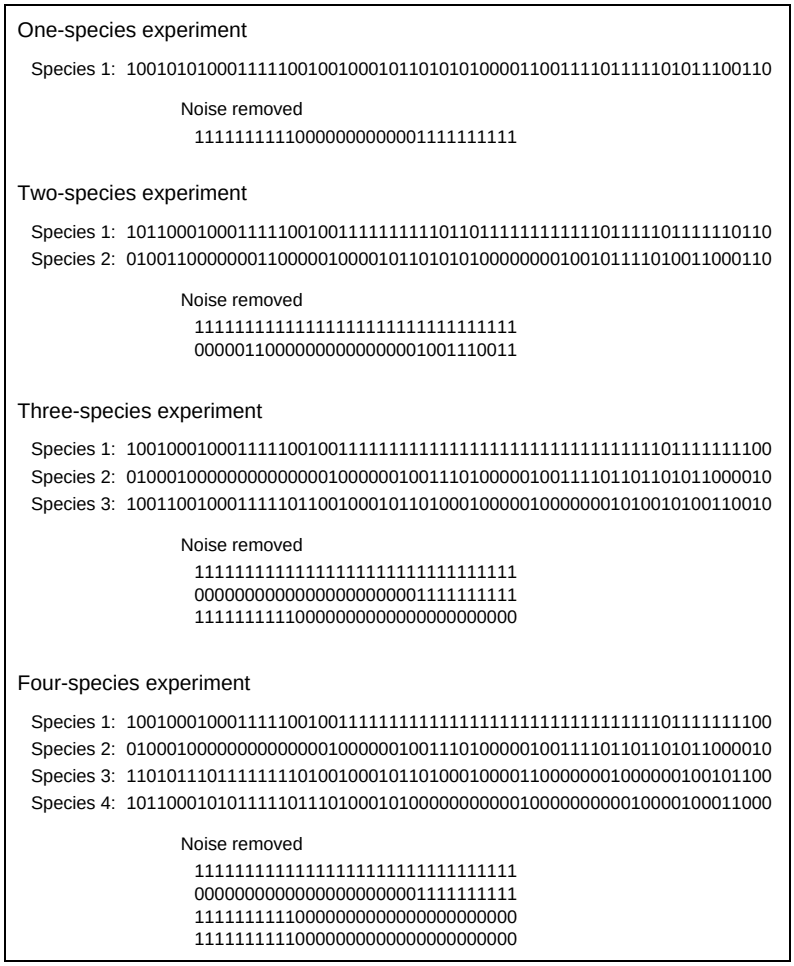

Figure 5: Final representatives from one through four species experiments before and after the removal bits corresponding to variable target regions.

strings. This enabled the ecosystem with four species to achieve a slightly higher match score on the full 64-bit target strings than the ecosystem with three species. To determine whether this difference is significant, an additional 95 runs were performed using the three- and four-species ecosystems to bring the total to 100 runs apiece. The arithmetic means from the two sets of runs were 51.037 and 51.258 respectively, and a t-test verified that this difference is unlikely to have occurred by chance.

4.2.3 Adaptation as the Number and Role of Species Change

If we do not know *a priori* how many species are required and must dynamically create them as evolution progresses, existing species must be able to adapt to these new species. One of the forces driving this adaptation is the freedom of older species to become more specialized as new species begin making useful contributions.

Specializing in this manner is similar to the notion of *character displacement* that occurs

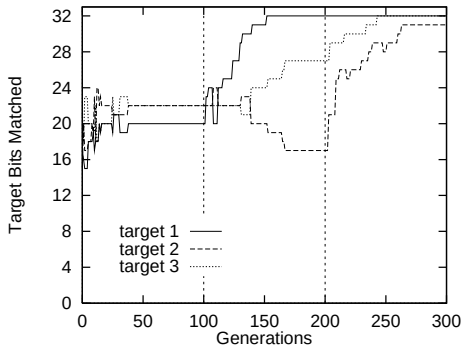

Figure 6: Shifting from generalists to specialists as new species are added to the ecosystem on a fixed schedule.

in natural ecosystems (Brown and Wilson, 1956). For example, a study of finches in the Galápagos by Lack (1947) determined that on the eight islands occupied by both *Geospiza fortis* and *Geospiza fuliginosa*, the average depth of the *G. fortis* beak was approximately 12 mm while the average depth of the *G. fuliginosa* beak was approximately 8 mm. However, on the islands Daphne, occupied only by *G. fortis*, and Crossman, occupied only by *G. fuliginosa*, the average beak depth of both species was approximately 10 mm. The interpretation of this observation is that when both finch species compete for food within the same ecosystem, their beaks evolve to become specialized to either a larger or smaller variety of seeds. However, when only one of these two species occupies an ecosystem, the species evolves a more general purpose beak suitable for consuming a wider variety of seeds.

To determine whether the species in our model of cooperative coevolution will mimic this process and become more specialized as new species are introduced into their ecosystem, we begin this experiment with a single species, add a second species at generation 100, and add a third species at generation 200. A 30-element target set was generated from the same three schemata used in Section 4.2.2.

The number of target bits from the fixed region of the schemata matched by the best individual from each species is shown in Figure 6. As before, although only the performance on the 32-bit fixed region is shown, the fitness of individuals was based on how well the entire 64 bits of each string was covered. Each dashed, vertical line marks the creation of a new species. A period of instability just after each species is introduced is evidence of a few quick role changes as the species "decide" which niche they will occupy. However, the roles of the species stabilize after they evolve for a few generations. It is clear from the figure that when the second species is introduced, one of the species specializes on the strings containing the first target pattern, while the other species generalizes to the strings containing the other two target patterns. Similarly, when the third species is introduced, all three species are able to become specialists. Furthermore, by comparing Figure 6 with the plots in Figure 4, we see that this adaptation results in the same levels of generalization and specialization that are achieved when the species are evolved together from the beginning.

4.2.4 Evolving an Appropriate Number of Species

It is important not to evolve too many species because each requires computational resources to support 1) the increasing number of fitness evaluations that need to be performed, 2) the

need for applying operators such as crossover and mutation to more individuals, and 3) the conversion between genotypic and higher-level representations that may be required. On the other hand, if we evolve too few species they will be forced to be very general—resulting in mediocre covers as we saw in the previous few studies.

An algorithm for evolving an appropriate number of species was introduced in Section 3.2. To test the effectiveness of this algorithm, we use the same target set as in the previous two studies and begin by evolving a single species. At each ecosystem generation[1] we check for evolutionary stagnation, and if we are not making sufficient improvement, we add a new randomly initialized species and destroy those that are not making a significant contribution. Precisely what constitutes stagnation is application dependent, but here we determined through experimentation that if the current-best fitness as computed from Equation 3 does not improve by at least 0.5 over five generations, further significant improvement without the addition of a new species is unlikely. Regarding species destruction, we consider that a species contributes to the fitness of a collaboration when its representative matches at least one of the target strings better than any other member of the collaboration, with ties being won by the older species. We refer to the amount of contribution that a species must make to be considered viable as its *extinction threshold*. We set the extinction threshold here to 5.0 to eliminate species that gain an advantage by matching spurious shared patterns in the random regions of the target strings. If we had been interested in these less significant patterns, we could have set the extinction threshold to a smaller value—perhaps just slightly above zero—and the species matching these patterns would be preserved.

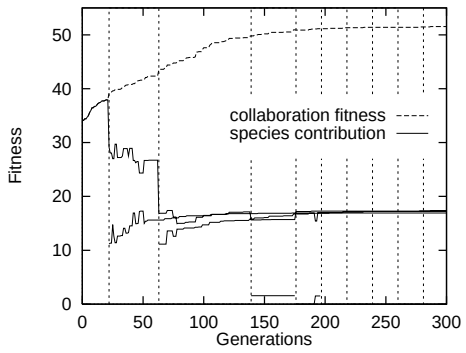

Figure 7: Changing contributions as species are dynamically created and eliminated from the ecosystem.

The fitness of the best collaboration and the amount of fitness contributed by each species in the ecosystem over 300 generations is plotted in Figure 7. The vertical, dashed lines represent stagnation events in which unproductive species are eliminated, and a new species is created. For the first 21 generations, the ecosystem consists of only one species; therefore, its contribution during this period is equal to the collaboration fitness. At generation 22 stagnation is detected, and a second species is added to the ecosystem. Although the contribution of the first species now drops from 37.9 to 29.1, the contribution of the new species causes the collaboration fitness to increase from 37.9 to 39.2. At generation 63 the evolution of these two species has stagnated, and we add a third species that

[1] By *ecosystem generation* we mean that all species are evolved for a generation.

focuses on some of the strings being matched by the first species—causing the contribution of the first species to drop to 16.8. However, the third species contributes 11.1, and the fitness of the collaboration jumps from 42.3 to 43.5. At generation 138 evolution has stagnated with three species in the ecosystem, the collaboration fitness is 49.7, and the species are contributing 17.1, 16.8, and 15.8 respectively. We know from the previous few studies that this is the optimal number of species for this particular problem; however, the system does not possess this knowledge and creates a new species. This species is only able to contribute 1.6 to the fitness of the collaboration, which is less than the extinction threshold, and, therefore, it is eliminated at generation 176. At this point another species is created, but it does not begin making a contribution until generation 192. Since the most this new species contributes is 1.5, it is eliminated at generation 197, and another new species is created. From this point until the end of the run, none of the new species ever makes a non-zero contribution, and each is eliminated in turn when stagnation is detected.

The first observation that can be made from this experiment is that an appropriate number of species emerges. Specifically, the ecosystem stabilizes in a state in which there are three species making significant contributions and a fourth exploratory species providing insurance against the contingency of a significant change in the environment. Our simple string covering application has a stationary objective function, so this insurance is not really necessary, but this would often not be the case in "real-world" problems. The second observation is that although the fourth and fifth species were eventually eliminated, they were able to make small contributions by matching patterns in the random regions of a few targets rather than generalizing over a large number of them. If we were interested in maintaining these specialists in the ecosystem we could lower the extinction threshold, and they would survive.

5 Case Study in Emergent Problem Decomposition

Moving beyond the simple studies of the previous section, we now describe a more complex case study in which cooperative coevolution is applied to the construction of an artificial neural network. Our task will be to construct a multilayered feed-forward artificial neural network that when presented with an input pattern, will produce some desired output signal. We are given the basic topology of the network, but we know neither the number of neural units required to adequately produce the desired output nor the role each unit should play—the determination of which constitutes our decomposition task. The primary focus of this case study is to directly compare and contrast the decompositions produced by cooperative coevolution to those produced by a non-evolutionary approach that is specific to the construction of artificial neural networks.

5.1 Evolving Cascade Networks

The topology evolved in this case study is a *cascade network*. In cascade networks, all input units have direct connections to all hidden units and to all output units, the hidden units are ordered, and each hidden unit sends its output to all downstream hidden units and to all output units. Cascade networks were originally used in conjunction with the cascade-correlation learning architecture (Fahlman and Lebiere, 1990).

Cascade network evolution can be easily mapped to the cooperative coevolution architecture by assigning one species to evolve the weights on the connections leading into the output units and assigning each of the other species to evolve the connection weights for one

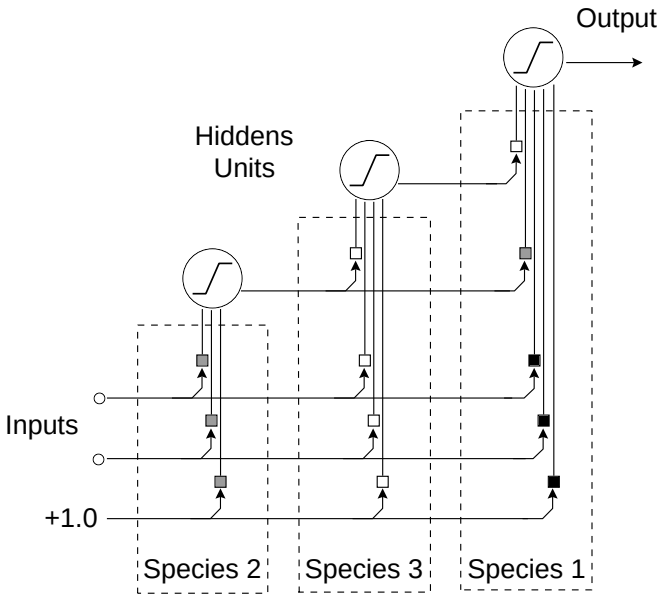

Figure 8: Mapping between cascade network and species.

of the hidden units.[2] For example, the species to network mapping for a cascade network having two inputs, a bias signal of 1.0, two hidden units, and one output is shown in Figure 8. Each of the small black, gray, and white boxes in the figure represents a connection weight being evolved. To add a new hidden unit to the network we simply create a new species for that unit and add a new randomly initialized weight to each of the individuals of the species assigned to the output unit. The fitness of an individual is determined by combining it with the current best individual from each of the other species to produce a connection weight specification for the entire network. The sum-squared error is then computed as preclassified training patterns are fed forward through the network. Individuals producing networks with lower sum-squared errors are considered more highly fit.

The network shown in Figure 8 is constructed incrementally as follows. When the evolution of the network begins, there is only one species in the ecosystem, and its individuals represent alternatives for the output connection weights denoted by the three black boxes. Later in the network's evolution, the first hidden unit is added, and a second species is created to represent the new unit's input connection weights. In addition, a new connection weight is added to each individual of the first species. All of these new weights are denoted by gray boxes in the figure. The species creation event is triggered by evolutionary stagnation as described earlier. Later still, evolution again stagnates and the second hidden unit is added, a third species is created to represent the unit's connection weights, and the individuals of the first species are further lengthened. These connection weights are denoted by white boxes. This cycle continues until a network is created that produces a sufficiently low sum-squared error. Using this scheme, a cascade network with k hidden units will be constructed from $k + 1$ species. Alternatively, we could have started with a liberal estimation of the number of hidden units required and let the destruction of species pare the network down to an

[2] Although we have not done so, other network topologies could also be easily mapped to the cooperative coevolution architecture.

appropriate size, but this option was not explored.

Due to the complexity of the neural network search space, when adding a new species, we found it helpful to temporarily focus computational resources on enabling it to find a niche in which it could make a contribution. This is accomplished simply by evolving just the new species and the species representing the weights on the connections to the output unit until progress again approaches an asymptote. At this point, we eliminate the new species if it is not making a significant contribution and create another one to replace it, or we continue with the evolution of *all* of the species in the ecosystem.

The EA used in this case study is a (μ, λ) evolution strategy (ES) as described by Schwefel (1995), with $\mu = 10$ and $\lambda = 100$. We previously applied a GA to this task (Potter and De Jong, 1995) but achieved better results with the ES. Each individual consists of two real-valued vectors: a vector of connection weights and a vector of standard deviations used by the mutation operator. We require the standard deviations to always be greater than 0.01 and they are adapted as follows:

$$\vec{\sigma}_{t+1} = \vec{\sigma}_t e^{Gauss(0,\sigma')} \tag{4}$$

where t denotes time expressed discretely in generations. The rate of convergence of the ES is sensitive to the choices of σ' and the initial setting of the standard-deviation vectors $\vec{\sigma}$. We follow Schwefel's recommendation and set σ' using the equation

$$\sigma' = \frac{C}{\sqrt{|\vec{\sigma}|}} \tag{5}$$

where C depends on μ and λ. Given the $(10, 100)$ ES used here, C is set to 1.0. Schwefel also recommends initializing $\vec{\sigma}$ using the equation

$$\sigma_k = \frac{R_k}{\sqrt{|\vec{\sigma}|}} \qquad \text{for } k = 1, 2, \ldots, |\vec{\sigma}| \tag{6}$$

where the constant R_k is the maximum uncertainty range of the corresponding variable. Given that our randomly initialized connection weights are limited to the range $(-10.0, 10.0)$, each R_k is correspondingly set to 20.0. Mutation is the only evolutionary operator used.

5.2 The Cascade-Correlation Approach to Decomposition

In the context of a cascade network, problem decomposition consists of determining the number of hidden units and their roles. We will be comparing and contrasting the decompositions produced by cooperative coevolution to those produced by a second-order gradient-descent technique for constructing cascade networks called the cascade-correlation learning architecture (Fahlman and Lebiere, 1990).

Prior to the development of the cascade-correlation learning architecture, feed-forward networks were constructed by using rules-of-thumb to determine the number of hidden layers, units, and their connectivity. The roles of the hidden units were allowed to emerge through the application of the backpropagation algorithm—a first-order gradient-descent technique (Rumelhart et al., 1986). A source of inefficiency in this process is the considerable time it takes for the emergence of roles to stabilize (Fahlman and Lebiere, 1990).

Cascade-correlation was specifically designed to eliminate this inefficiency by constructing the network one hidden unit at a time and freezing the roles of the hidden units once established. Rather than simply allowing these roles to emerge, each hidden unit is trained to respond either positively or negatively to the largest portion of remaining error signal using gradient-descent to adjust its input connection weights[3]. The gradient is with respect to the magnitude of the correlation between the output from the hidden unit and the sum-squared error as training patterns are fed forward through the network. After training, the hidden unit will only fire when the most problematic patterns from the training set are presented to the network—forcing the hidden unit to focus on a specific region of the input space. Once the input connection weights of a hidden unit are trained and frozen, all output connection weights are trained by descending the sum-squared network error gradient. The addition of a new hidden unit is triggered when the reduction of the sum-squared network error approaches an asymptote. This cycle of adding a new hidden unit, training and freezing its input connection weights, and training all output connection weights continues until a sufficiently low sum-squared error is produced.

5.3 Two-Spirals Problem

We will construct cascade networks to solve the two-spirals problem originally proposed in a post to the *connectionists* mailing list by Alexis Wieland. This problem is a classification task that consists of deciding in which of two interlocking spiral-shaped regions a given (x, y) coordinate lies. The interlocking spiral shapes were chosen for this problem because they are not linearly separable. Finding a neural network solution to the two-spirals problem has proven to be very difficult when using a traditional gradient-descent learning method such as backpropagation, and, therefore, it has been used in a number of studies to test new learning methods (Lang and Witbrock, 1988; Fahlman and Lebiere, 1990; Whitley and Karunanithi, 1991; Suewatanakul and Himmelblau, 1992; Potter, 1992; Karunanithi et al., 1992).

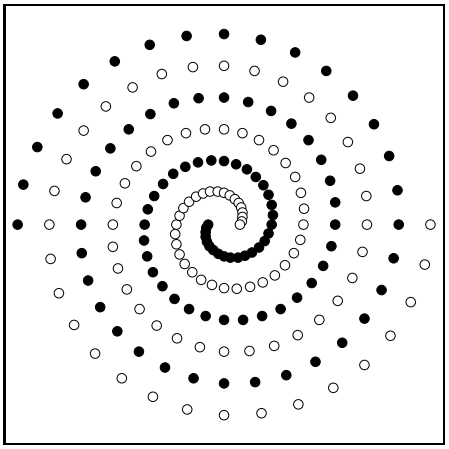

Figure 9: Training set for the two-spirals problem.

To learn to solve this task, we are given a training set consisting of 194 preclassified

[3] More accurately, a small number of *candidate units* are created and trained in parallel; however, for the sake of clarity, we ignore that detail in this description.

coordinates as shown in Figure 9. Half of the coordinates are located in one spiral-shaped region and marked with black circles, and half are in the other spiral-shaped region and marked with white circles. The coordinates of the 97 black circles are generated using the following equations:

$$r = \frac{6.5(104 - i)}{104} \quad (7)$$

$$\theta = i\frac{\pi}{16} \quad (8)$$

$$x = r\sin\theta \quad (9)$$

$$y = r\cos\theta \quad (10)$$

where $i = 0, 1, \ldots, 96$. The coordinates of the white circles are generated simply by negating the coordinates of the black circles.

When performing a correct classification, the neural network takes two inputs corresponding to an (x, y) coordinate and produces a positive signal if the point falls within the black spiral and a negative signal if the point falls within the white spiral.

5.4 Experimental Results

Ten runs were performed using cascade-correlation, and ten runs were performed using cooperative coevolution. Runs were terminated when either all 194 training patterns were classified correctly, or it was subjectively determined that learning was stuck at a local optimum. Cascade-correlation generated networks capable of correctly classifying all the training patterns in all ten runs, while cooperative coevolution was successful in seven out of ten runs. We will address the three unsuccessful runs at the end of this section.

Table 1: Required number of hidden units.

Method	*Hidden units*		
	Mean	*Max*	*Min*
Cascade-correlation	16.80 ± 1.16	19	14
Cooperative coevolution	13.71 ± 2.18	18	12

Table 1 shows the average number of hidden units produced by each method, along with 95-percent confidence intervals on the mean, and the maximum and minimum number of hidden nodes generated. Cascade-correlation networks required an average of 16.8 hidden units, which is consistent with results reported by Fahlman and Lebiere (1990). In contrast, in its seven successful runs, cooperative coevolution networks required an average of only 13.7 hidden units to perform the task. This represents a statistically significant difference in the number of hidden units required to solve the problem. The *p*-value produced from a *t*-test of the means was 0.015.

We begin our characterization of the roles played by the hidden units produced by the two methods by describing the reduction in misclassification and sum-squared error attributable to each unit. Table 2 was generated by starting with the final networks produced by the first runs of the two methods and eliminating one hidden unit at a time while measuring the number of training-set misclassifications and the sum-squared error.

Table 2: Effect of adding hidden units on training set classification.

Hidden units	*Misclassifications*		*Sum-squared error*	
	CasCorr	*CoopCoev*	*CasCorr*	*CoopCoev*
0	96	99	84.96	68.26
1	94	97	83.41	64.96
2	76	84	64.61	61.34
3	74	70	64.68	67.24
4	64	80	62.21	68.36
5	64	72	61.45	54.57
6	58	70	50.65	62.53
7	54	67	37.98	54.76
8	58	44	46.24	35.38
9	52	61	35.04	46.84
10	36	27	30.27	20.78
11	34	27	25.38	17.18
12	26	0	21.52	6.63
13	22		14.49	
14	16		8.87	
15	0		1.67	

The first run was chosen for this comparison arbitrarily; however, it appears to provide a reasonably fair comparison. Overall, we find the sequences from the two methods to be quite similar. One similarity is that neither the misclassification nor the sum-squared error sequences monotonically decrease; that is, both methods have created hidden units that, when looked at in isolation, make matters worse. These units presumably play more complex roles—perhaps working in conjunction with other hidden units. Another similarity is that the misclassification sequences of both methods are more erratic than the sum-squared error sequences; however, this is no surprise because neither method used misclassification information for training. The major difference between the methods is that the cooperative coevolution sequences tend to make bigger steps with higher variance and contain fewer elements.

As in Fahlman and Lebiere (1990), we gain further insight into the roles played by the hidden units by studying a series of field-response diagrams generated from the same networks summarized in Table 2. The field-response diagrams shown in Figure 10 were produced from the cascade-correlation network, and those shown in Figure 11 were produced from the network evolved with cooperative coevolution. The diagrams were generated by feeding the elements of a 256×256 grid of coordinates forward through the network and measuring the output signal produced both by individual hidden units and the entire network. Positive signals are displayed as black pixels, and negative signals are displayed as white pixels. For example, in Figure 10 the top-right pair of field response diagrams is generated from a cascade-correlation network in which all but the first two hidden units have been eliminated. The left diagram of that particular pair shows the output from the second hidden unit, and the right diagram of the pair shows the corresponding output from the network.

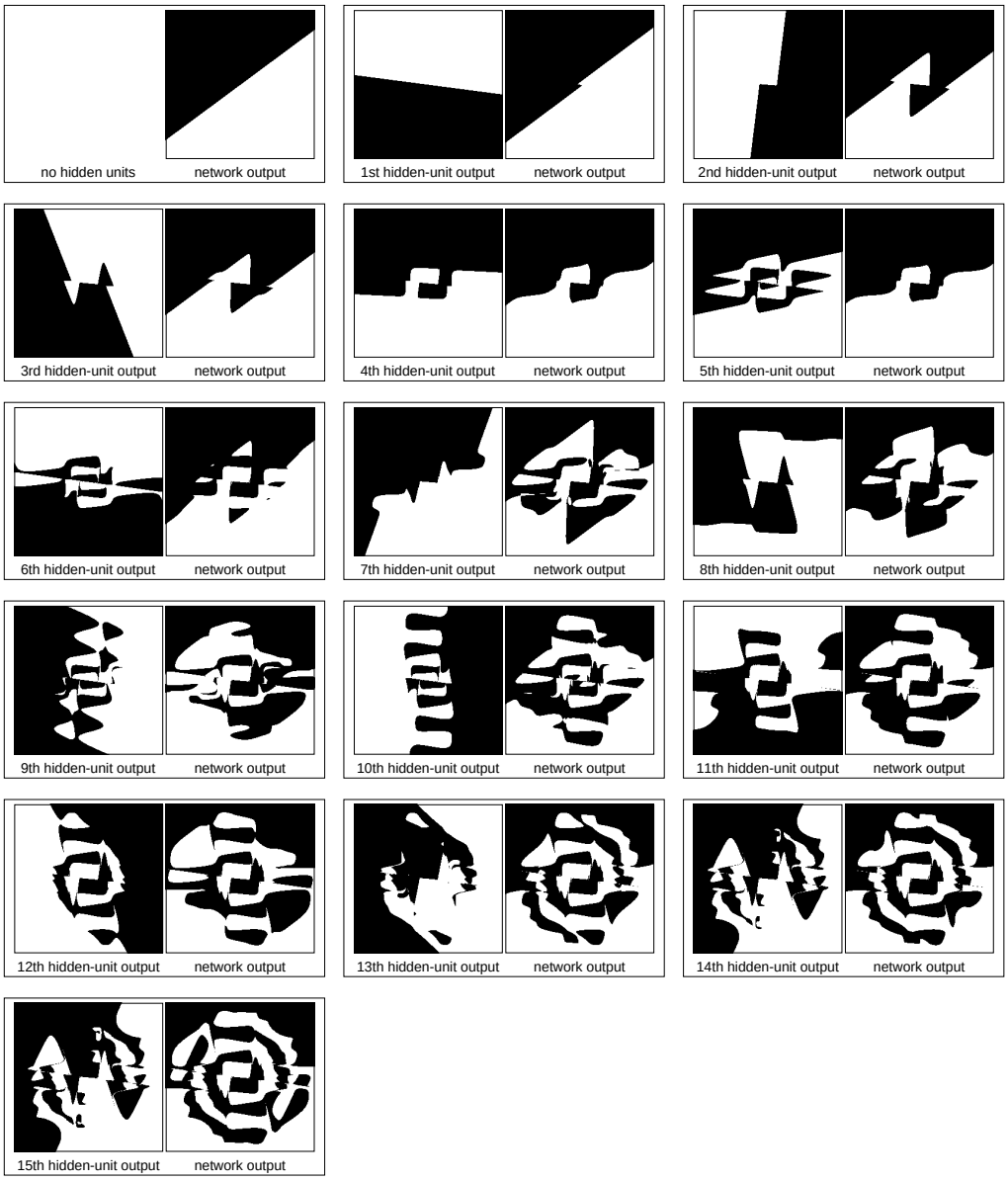

Figure 10: Field response diagrams generated from a neural network constructed using cascade correlation, showing the incremental effect of adding hidden units to the network.

We make a number of observations from a comparison of these two sets of figures. First, both the cascade-correlation decompositions and those produced by cooperative coevolution clearly exploit the symmetry inherent in the two-spirals problem. A second similarity is that the early hidden units focus on recognition in the center region of the field. This shows that both methods are exploiting the greater central concentration of training set elements, as one can see from Figure 9. A third similarity is that as hidden units are added to the network, their response patterns tend to become increasingly complex;

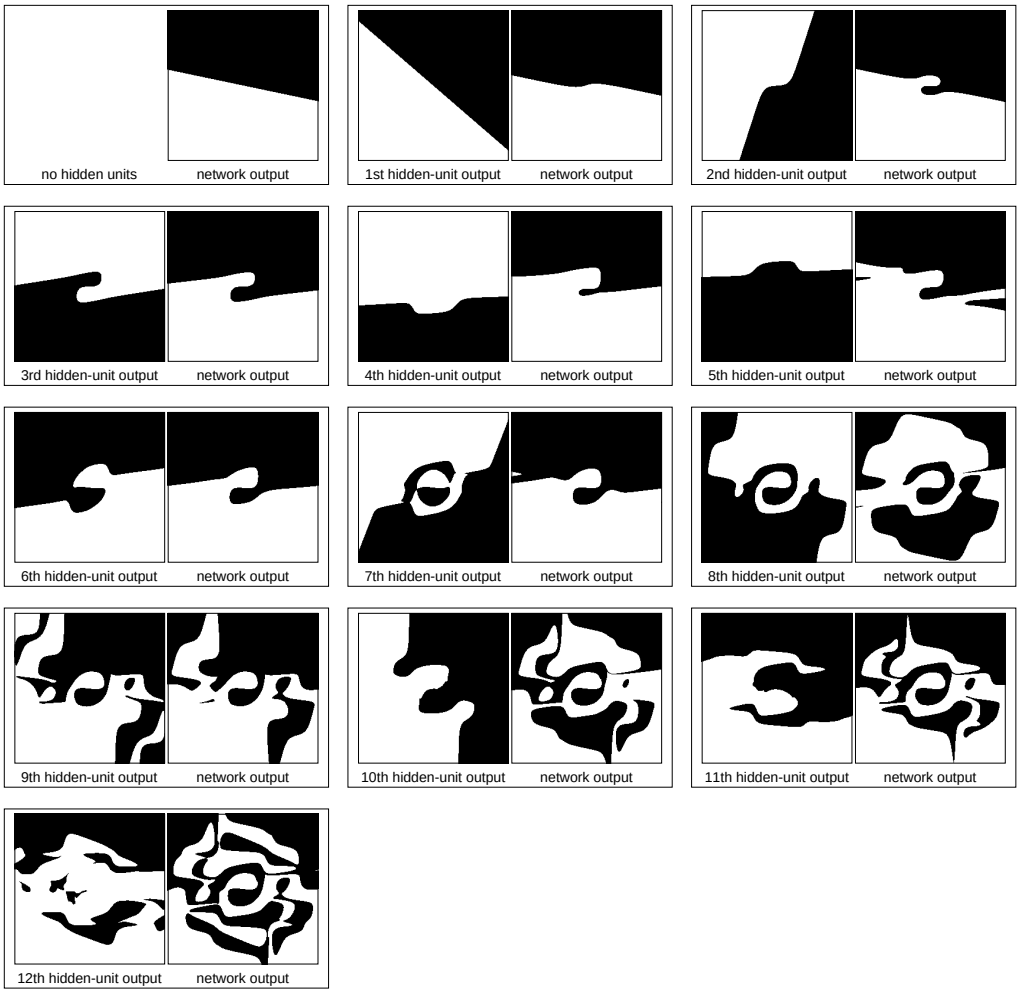

Figure 11: Field response diagrams generated from a neural network constructed using cooperative coevolution, showing the incremental effect of adding hidden units to the network.

although, this is less true with cooperative coevolution than with cascade-correlation. The increase in complexity may simply be a result of the network topology—the later hidden units have more inputs than the early hidden units as shown in Figure 8.

There are also a couple of noticeable differences between the two sets of figures. The cascade-correlation field-response diagrams tend to consist of angular-shaped regions, while the shapes in the diagrams produced by the network evolved with cooperative coevolution are more rounded. In addition, the cascade-correlation diagrams are visually more complex than the ones from cooperative coevolution. We hypothesize that these differences between the decompositions, as highlighted by the field-response diagrams, are due to the task-specific nature of the cascade-correlation decomposition technique. Recall that cascade-correlation uses the correlation between the output of a hidden unit and the network error signal to train the weights on the connections leading into the unit. This enables the hidden

unit to respond precisely to the training patterns that are responsible for most of the error signal while ignoring the other training patterns. This is manifested in the field-response diagrams as complex angular regions. Since this particular instantiation of the cooperative coevolution architecture does not use task-specific statistical information as a focusing tool, it tends to paint with broader brush strokes.

Painting with broad brush strokes may be advantageous with respect to the ability to generalize, but it can be a disadvantage as well. In all three unsuccessful runs, after all but a few of the training patterns were correctly classified, the last species created was unable to find a niche in which it could contribute. Further investigation of the unsuccessful runs revealed that the sum-squared error generated by the few remaining misclassifications was being masked by the residual sum-squared error generated by all the other training patterns. Therefore, variation among individuals produced no selective advantage, and, hence, no further evolutionary progress occurred. This could probably be addressed by utilizing the correlation statistic in the fitness function of the species representing hidden units.

In summary, our approach to letting decompositions emerge purely as a result of evolutionary pressure to increase the overall fitness of the ecosystem works well in simple domains, and it even has the potential of producing good decompositions in domains as complex as the one described here. However, as our domains become increasingly complex we may need to bestow the fitness functions with a more specialized ability to steer the species towards niches in which they can make useful contributions.

6 Conclusions and Future Work

We believe that to apply evolutionary algorithms to the solution of increasingly complex problems, explicit notions of modularity must be introduced to provide reasonable opportunities for solutions to evolve in the form of interacting coadapted subcomponents. The difficulty comes in finding computational extensions to our current evolutionary paradigms in which such subcomponents "emerge" rather than being hand designed. In this paper we have described an architecture for evolving such subcomponents as a collection of cooperating species. We showed that evolutionary pressure to increase the overall fitness of the ecosystem can provide the needed stimulus for the emergence of an appropriate number of interdependent subcomponents that cover multiple niches, evolve to an appropriate level of generality, and adapt as the number and roles of their fellow subcomponents change over time. Finally, we explored the emergence of coadapted subcomponents in more detail through a case study involving neural network evolution, which suggested that, as our domains become increasingly complex, the evolution of species may need to be driven by more than the overall fitness of the ecosystem to produce good decompositions.

The results are quite positive in a number of ways. First, the architecture has demonstrated the ability to evolve useful decompositions expressed computationally as the emergence of cooperating species. Second, the cooperative coevolutionary architecture has been shown to be a general extension for any of the standard EC paradigms and not tightly coupled to a specific approach such as GAs. Finally, this architecture is allowing us to scale up to much more complex problems than possible with our standard EAs. As an example, we are currently using this approach to evolve complex behaviors for robots in multi-agent environments. In this regard, the architecture is suitable both for coevolving various modalities of behavior within a single robot, and for coevolving the behavior of a group of cooperating robots.

At the same time there are a number of improvements to the current system that we are exploring. At present, the mechanism for evaluating the collaborative potential of coevolving components has been deliberately kept as simple as possible. There are several ways in which such evaluations could be made to be more effective. Similarly, the mechanism for the creation and destruction of species is currently quite simplistic and provides opportunities for significant improvements. Finally, our current coevolutionary architecture is implemented serially even though parallel implementations are quite natural. Of particular interest as we scale up to more complex domains are implementations of the architecture in which species are assigned to separate processors and coevolved asynchronously.

Acknowledgments

This work was supported by the Office of Naval Research.

References

Brown, Jr., W. L. and Wilson, E. O. (1956). Character displacement. *Systematic Zoology*, 5(2):49–64.

de Garis, H. (1990). Building artificial nervous systems using genetically programmed neural network modules. In Porter, B. and Mooney, R., editors, *Proceedings of the Seventh International Conference on Machine Learning*, pages 132–139, Morgan Kaufmann, Palo Alto, California.

Fahlman, S. E. and Lebiere, C. (1990). The cascade-correlation learning architecture. Technical Report CMU-CS-90-100, School of Computer Science, Carnegie Mellon University, Pittsburgh, Pennsylvania.

Forrest, S. and Perelson, A. S. (1990). Genetic algorithms and the immune system. In Schwefel, H.-P. and Männer, R., editors, *Parallel Problem Solving from Nature*, pages 320–325, Springer-Verlag Berlin, Germany.

Forrest, S., Javornik, B., Smith, R. E. and Perelson, A. S. (1993). Using genetic algorithms to explore pattern recognition in the immune system. *Evolutionary Computation*, 1(3):191–211.

Giordana, A. and Neri, F. (1996). Search-intensive concept induction. *Evolutionary Computation*, 3(4):375–416.

Giordana, A., Saitta, L. and Zini, F. (1994). Learning disjunctive concepts by means of genetic algorithms. In Cohen, W. and Hirsh, H., editors, *Proceedings of the Eleventh International Conference on Machine Learning*, pages 96–104, Morgan Kaufmann, San Francisco, California.

Hillis, D. W. (1991). Co-evolving parasites improve simulated evolution as an optimization procedure. In Langton, C. G., Taylor, C., Farmer, J. D. and Rasmussen, S., editors, *Artificial Life II, SFI Studies in the Sciences of Complexity*, Volume 10, pages 313–324. Addison-Wesley, Redwood City, California.

Holland, J. H. (1986). Escaping brittleness: The possibilities of general purpose learning algorithms applied to parallel rule-based systems. In Michalski, R. S., Carbonell, J. G. and Mitchell, T. M., editors, *Machine Learning*, Volume 2, pages 593–623. Morgan Kaufman, Los Altos, California.

Holland, J. H. and Reitman, J. S. (1978). Cognitive systems based on adaptive algorithms. In Waterman, D. A. and Hayes-Roth, F., editors, *Pattern-Directed Inference Systems*. Academic Press, New York, New York.

Husbands, P. and Mill, F. (1991). Simulated co-evolution as the mechanism for emergent planning and scheduling. In Belew, R. K. and Booker, L. B., editors, *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 264–270, Morgan Kaufmann, San Mateo, California.

Karunanithi, N., Das, R. and Whitley, D. (1992). Genetic cascade learning for neural networks. In Whitley, L. D. and Schaffer, J. D., editors, *COGANN-92 International Workshop on Combinations of Genetic Algorithms and Neural Networks*, pages 134–145, IEEE Computer Society Press, Piscataway, New Jersey.

Kauffman, S. A. and Johnsen, S. (1991). Co-evolution to the edge of chaos: Coupled fitness landscapes, poised states, and co-evolutionary avalanches. In Langton, C. G., Taylor, C., Farmer, J. D. and Rasmussen, S., editors, *Artificial Life II, SFI Studies in the Sciences of Complexity*, Volume 10, pages 325–369. Addison-Wesley, Redwood City, California.

Koza, J. R. (1993). Hierarchical automatic function definition in genetic programming. In Whitley, L. D., editor, *Foundations of Genetic Algorithms 2*, pages 297–318. Morgan Kaufmann, San Mateo, California.

Lack, D. L. (1947). *Darwin's Finches*. Cambridge University Press, Cambridge, England.

Lang, K. J. and Witbrock, M. J. (1988). Learning to tell two spirals apart. In Touretzky, D., Hinton, G. and Sejnowski, T., editors, *Proceedings of the 1988 Connectionist Models Summer School*, pages 52–59, Morgan Kaufmann, San Mateo, California.

Lin, L.-J. (1993). Hierarchical learning of robot skills by reinforcement. In *Proceedings of the 1993 International Joint Conference on Neural Networks*, pages 181–186, IEEE Computer Society Press, Piscataway, New Jersey.

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

Moriarty, D. E. and Miikkulainen, R. (1997). Forming neural networks through efficient and adaptive coevolution. *Evolutionary Computation*, 5(4):373–399.

Paredis, J. (1995). The symbiotic evolution of solutions and their representations. In Eshelman, L., editor, *Proceedings of the Sixth International Conference on Genetic Algorithms*, pages 359–365, Morgan Kaufmann, San Francisco, California.

Potter, M. A. (1992). A genetic cascade-correlation learning algorithm. In Whitley, L. D. and Schaffer, J. D., editors, *COGANN-92 International Workshop on Combinations of Genetic Algorithms and Neural Networks*, pages 123–133, IEEE Computer Society Press, Piscataway, New Jersey.

Potter, M. A. and De Jong, K. A. (1994). A cooperative coevolutionary approach to function optimization. In Davidor, Y. and Schwefel, H.-P., editors, *Proceedings of the Third Conference on Parallel Problem Solving from Nature*, pages 249–257, Springer-Verlag, Berlin, Germany.

Potter, M. A. and De Jong, K. A. (1995). Evolving neural networks with collaborative species. In Ören, T. I. and Birta, L. G., editors, *Proceedings of the 1995 Summer Computer Simulation Conference*, pages 340–345, The Society for Computer Simulation, San Diego, California.

Potter, M. A., De Jong, K. A. and Grefenstette, J. J. (1995). A coevolutionary approach to learning sequential decision rules. In Eshelman, L., editor, *Proceedings of the Sixth International Conference on Genetic Algorithms*, pages 366–372, Morgan Kaufmann, San Francisco, California.

Rosca, J. P. and Ballard, D. H. (1994). Hierarchical self-organization in genetic programming. In Cohen, W. and Hirsh, H., editors, *Proceedings of the Eleventh International Conference on Machine Learning*, pages 251–258, Morgan Kaufmann, San Francisco, California.

Rosca, J. P. and Ballard, D. H. (1996). Discovery of subroutines in genetic programming. In Angeline, P. and Kinnear, K. E., editors, *Advances in Genetic Programming 2*, Chapter 9. The MIT Press, Cambridge, Massachusetts.

Rosin, C. D. and Belew, R. K. (1995). Methods for competitive co-evolution: Finding opponents worth beating. In Eshelman, L., editor, *Proceedings of the Sixth International Conference on Genetic Algorithms*, pages 373–380, Morgan Kaufmann, San Francisco, California.

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

Samuel, A. L. (1959). Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development*, 3(3):210–229.

Schwefel, H.-P. (1995). *Evolution and Optimum Seeking*. John Wiley and Sons, New York, New York.

Singh, S. P. (1992). Transfer of learning by composing solutions of elemental sequential tasks. *Machine Learning*, 8:323–339.

Skinner, B. F. (1938). *The Behavior of Organisms: An Experimental Analysis.* Appleton-Century, New York, New York.

Smith, J. M. (1989). *Evolutionary Genetics*. Oxford University Press, New York, New York.

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

Suewatanakul, W. and Himmelblau, D. M. (1992). Comparison of artificial neural networks and traditional classifiers via the two-spiral problem. In Padgett, M. L., editor, *Proceedings of the Third Workshop on Neural Networks: Academic/Industrial/NASA/Defense*, pages 275–282, Society for Computer Simulation, San Diego, California.

Whitley, D. and Karunanithi, N. (1991). Generalization in feed forward neural networks. In *Proceedings of the International Joint Conference on Neural Networks – Seattle*, Volume 2, pages 77–82, IEEE, Piscataway, New Jersey.

Wright, S. (1932). The roles of mutation, inbreeding, crossbreeding and selection in evolution. In Jones, D. F., editor, *Proceedings of the Sixth International Conference of Genetics*, pages 356–366, Brooklyn Botanic Garden, Brooklyn, New York.