The Design and Analysis of a Computational Model of Cooperative Coevolution

A dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy at George Mason University

by

Mitchell A. Potter

BA, Mathematics, Florida State University, 1978
MS, Computer Science, University of Texas at Austin, 1985

Director: Kenneth A. De Jong, Associate Professor
Department of Computer Science

Spring Semester 1997
George Mason University
Fairfax, Virginia

Copyright 1997 Mitchell A. Potter
All Rights Reserved

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

DISSERTATION ABSTRACT

THE DESIGN AND ANALYSIS OF A COMPUTATIONAL MODEL OF COOPERATIVE COEVOLUTION

Mitchell A. Potter
George Mason University, 1997
Thesis Director: Dr. Kenneth A. De Jong

As evolutionary algorithms are applied to the solution of increasingly complex systems, 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. 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.

We begin by describing a computational model of cooperative coevolution that includes the explicit notion of modularity needed to provide reasonable opportunities for solutions to evolve in the form of interacting coadapted subcomponents. In this novel approach, subcomponents are represented as genetically isolated species and evolved in parallel. Individuals from each species temporarily enter into collaborations with members of the other species and are rewarded based on the success of the collaborations in solving objective functions.

Next, we perform a sensitivity analysis on a number of characteristics of decomposable problems likely to have an impact on the effectiveness of the coevolutionary model. Through focused experimentation using tunable test problems chosen specifically to measure the effect of these characteristics, we provide insight into their influence and how any exposed difficulties may be overcome.

This is followed by a study of the basic problem-decomposition capability of the model. We show, within the context of a relatively simple environment, that evolutionary pressure can provide the needed stimulus for the emergence of an appropriate number of subcomponents that cover multiple niches, are evolved to an appropriate level of generality, and can adapt to a changing environment. We also perform two case studies in emergent decomposition on complex problems from the domains of artificial neural networks and concept learning. These case studies validate the ability of the model to handle problems only decomposable into subtasks with complex and difficult to understand interdependencies.

Chapter 1

INTRODUCTION

For over three decades we have been applying the basic principles of evolution to the solution of technical problems in a variety of domains. This work was begun independently by Rechenberg (1964) with his work on *Evolutionsstrategie* for function optimization, Fogel (1966) with the evolution of finite state machines through *Evolutionary Programming*, and Holland (1975) with a class of adaptive systems we now call *Genetic Algorithms.*

The fundamental principles on which all of these computational models of evolution are based can best be summarized by nineteenth century naturalist Charles Darwin. In his introduction to *The Origin of Species*, Darwin (1859) makes the following observation:

> As many more individuals of each species are born than can possibly survive; and as, consequently, there is a frequently recurring struggle for existence, it follows that any being, if it vary however slightly in any manner profitable to itself, under the complex and sometimes varying conditions of life, will have a better chance of surviving, and thus be *naturally selected.* From the strong principle of inheritance, any selected variety will tend to propagate its new and modified form.

When we apply these principles to the solution of technical problems through computer simulation, the individuals Darwin refers to become alternative solutions to the problem of interest. The frequently recurring struggle for existence is simulated by limiting the size of the population of problem solutions stored in computer memory. Variation between individuals results from making random changes to the population of evolving solutions by mutating them in some fashion, and from recombining pieces of old solutions to produce new solutions. The process of natural selection, in which profitable variations increase the likelihood of endowed individuals surviving and passing their characteristics on to future generations, can be modeled in a number of ways. Some evolutionary algorithms, for example, use a technique in which better problem solutions have a higher probability of being recombined into new solutions and thereby preserving the attributes that made them viable. Other evolutionary algorithms take the opposite approach by ensuring that the poorer solutions have a higher probability of being eliminated from the population.

1.1 Motivation

Computational models of evolution have a number of advantages over other problem-solving methods. First, they can be applied when one has only limited knowledge about the problem being solved. For example, unlike some other problem-solving methods, the application of an evolutionary algorithm to a function optimization problem would not require knowledge of first or second derivatives, discontinuities, and so on. The minimal requirement is the ability to approximate the relative worth of alternative problem solutions. In the field of evolutionary computation, we use the term *fitness* to mean the relative worth of a solution as defined by some objective function. Note that this definition differs somewhat from the meaning of fitness in population genetics, which is the expected frequency of a particular genotype in the population. Second, evolutionary computation is less susceptible to becoming trapped by local optima. This is because evolutionary algorithms maintain a population of alternative solutions and strike a balance between exploiting regions of the search space that have previously produced fit individuals and continuing to explore uncharted territory. Third, evolutionary computation can be applied in the context of noisy or non-stationary objective functions. This advantage makes the evolutionary computation model attractive for solving problems in a wide range of domains, particularly when the goal is to construct a system that exhibits some of the characteristics of biology, such as intelligence or the ability to adapt to change.

At the same time, difficulties can and do arise in applying the traditional computational models of evolution to some classes of problems. We are particularly interested in three such problem classes. The first class includes problems in which multiple distinct solutions are required, as in multimodal function optimization. The second class is composed of problems in which many small specialized subcomponents are required to form a composite solution, as in rule-based systems and artificial neural networks. Problems from this class are sometimes referred to as *covering problems* due to their similarity to the classic set covering problem from mathematics. The third class consists of problems that are decomposable into a number of simpler subtasks and can most effectively be solved using a *divide-and-conquer* strategy, as would be the case, for example, in route planning tasks such as the traveling salesman problem, which can be decomposed into simpler subtours, and in behavior learning tasks such as those one would encounter in the domain of robotics, where complex behavior can be decomposed into simpler subbehaviors.

There are two primary reasons traditional evolutionary algorithms have difficulties with these types of problems. First, the population of individuals evolved by these algorithms has a strong tendency to converge because an increasing number of trials are allocated to observed regions of the solution space with above average fitness. This is a major disadvantage when solving multimodal function optimization problems where the solution needs to provide more information than the location of a single peak or valley. This strong convergence property also precludes the long-term preservation of coadapted subcomponents required for solving covering problems or utilizing the divide-and-conquer strategy, because any but the strongest individual will ultimately be eliminated. Second, individuals evolved by traditional evolutionary algorithms typically represent complete solutions and are evaluated in isolation. Since interactions between population members are not modeled, even if population diversity were somehow preserved, the evolutionary model would have to be extended to enable coadaptive behavior to emerge.

The hypothesis underlying this dissertation is that to apply evolutionary algorithms effectively to increasingly difficult 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 reasonable computational extensions to our current evolutionary paradigms in which such 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.

1.2 Current Approaches

One of the earliest examples of extending the basic evolutionary model to allow coadapted subcomponents to emerge is Holland's (1978) *classifier system* for rule learning. Classifier systems attempt to accomplish this by way of a single population of interacting rules whose individual fitness values are determined by their interactions with other rules through a simulated micro-economy. Other extensions have been proposed to encourage the emergence of niches and species in a single population, for example, De Jong's (1975) *crowding* technique and the *fitness sharing* technique of Goldberg (1987).

The use of multiple interacting subpopulations has also been explored as an alternative mechanism for the emergence of niches using the so-called *island model*. In the island model a fixed number of subpopulations (breeding islands) evolve competing solutions. In addition, individuals occasionally migrate from one island to another, so there is a gradual mixing of genetic material. The work of Grosso (1985) represents an early example of extending an evolutionary algorithm using the island model. Some previous work also has looked at cooperating and competing genetically isolated subpopulations, for example, the work on emergent planning and scheduling by Husbands (1991), and the coevolution of parasites and hosts by Hillis (1991).

These previous approaches suffer from a number of limitations. Classifier systems are complex and are limited to the evolution of rule-based systems. The extensions for the emergence of niches in a single population, such as crowding and fitness sharing, are appropriate for multimodal function optimization but do not model the interaction between subcomponents required when solving covering problems or using the divide-and-conquer strategy. Similarly, the island model has been used primarily to slow convergence and does not support the type of interaction between subcomponents required for them to coadapt and form competitive, exploitative, or cooperative relationships. Regarding the previous coevolutionary approaches, such as the parasites and hosts model of Hillis, they support rich interactions between species but have involved a user-specified decomposition of the problem.

1.3 Objectives

The primary goal of this dissertation is to develop a new macroevolutionary model of *cooperative coevolution* that combines and extends ideas from earlier evolutionary approaches to improve their generality and their ability to evolve interacting coadapted subcomponents without human involvement. The following objectives are our milestones in achieving this goal:

- To design and implement a computational model of cooperative coevolution that includes the explicit notion of modularity needed to provide reasonable opportunities for solutions to evolve in the form of interacting coadapted subcomponents.
- To study the effect of some important characteristics of decomposable problems on the performance of the coevolutionary model.
- To analyze the emergent problem-decomposition capabilities of the coevolutionary model.
- To apply the coevolutionary model to problems that can only be decomposed into subtasks with complex and difficult to understand interdependencies, and compare and contrast the resulting decompositions with those produced by task-specific non-evolutionary methods.

1.4 Methodology

We will primarily use an experimental methodology backed up with statistical analysis to achieve the objectives of this dissertation. In cases where it is possible to measure statistical significance, plots showing the mean performance over time will be overlaid with 95-percent confidence intervals computed from Student's (1908) t-statistic. We use 95-percent confidence intervals rather than the more common standard-error bars because confidence intervals provide a more precise measure of the likely true mean (Miller 1986). We will also compute p-values from the two-sample t-test or an analysis of variance when appropriate. For a detailed description of these tests see a basic book on statistical analysis, for example, (Bhattacharyya and Johnson 1977). We generally check our distributions for normality; however, Miller (1986) shows that as long as the sample sizes are sufficiently large the t-statistic is robust for validity even when the distributions deviate from normal.

Performance comparisons are not made with non-evolutionary methods other than in the emergent decomposition case studies. The complex issue of whether the basic evolutionary paradigm is "better" than other methods from a computational perspective on a particular class of problems is not the focus of this dissertation. The interested reader is referred to Schwefel (1995) for a detailed comparison of evolutionary computation and traditional function optimization techniques, Schaffer et al. (1992) for a survey of a number of studies comparing evolutionary algorithms with gradient methods for training neural networks, and Neri and Saitta (1996) for a comparison of genetic search and symbolic methods for concept learning.

1.5 Proposed Coevolutionary Model

In the proposed cooperative coevolutionary model, multiple instances of an evolutionary algorithm are run in parallel; each instance of which evolves a genetically isolated population of interbreeding individuals. Because only individuals within the same population have the potential of mating, by definition, each population represents a single *species*. Although the species are isolated genetically, they are evaluated within the context of each other. Specifically, each species will enter into a temporary collaboration with members of the other species and will be rewarded based on the success of the collaboration. Therefore the species are considered *sympatric*, that is, they live in the same place, rather than *allopatric*, meaning geographically isolated, and their ecological relationship is one of helping each other, which is referred to in the field of evolutionary genetics as *mutualism* (Smith 1989).

The proposed coevolutionary model has a number of beneficial characteristics. First, it is a general problem solving method that is applicable to a variety of problem classes. For example, in the chapters that follow we will apply the model to string covering, function optimization, concept learning, and the evolution of neural networks. Second, it is a macroevolutionary model that is not limited to a particular underlying evolutionary algorithm. We will show, for example, that it can extend the usefulness of both genetic algorithms and evolution strategies. Third, the model is efficient. The evolution of genetically isolated species in separate populations can be easily distributed across a network of processors with little communication overhead and unproductive cross-species mating is eliminated. In addition, by evaluating individuals from one species within the context of individuals from other species, the model constrains the search space in a fashion similar to the coordinate strategy used in traditional parameter optimization; see, for example, (Schwefel 1995, 41–44). This enables high-dimensional problems to be solved more efficiently. Fourth, the dynamics of the model are such that reasonable problem decompositions emerge due to evolutionary pressure rather than being specified by the user.

1.6 Contributions

The main contributions of this dissertation are as follows:

- We have designed and implemented a novel computational model of cooperative coevolution in which the subcomponents of a problem solution are drawn from a collection of genetically isolated species that collaborate with one another to achieve a common goal.

- We have performed a sensitivity analysis on four characteristics of decomposable problems likely to have a major impact on the performance of the coevolutionary model. Specifically, we have analyzed the effect of the amount and structure of interdependency between problem subcomponents, the dimensionality of the decomposition, and the ability of the model to handle inaccuracy in the fitness evaluation of the collaborations.

- We have shown that evolutionary pressure can be a powerful force in provoking the emergence of coadapted species that, working together, are able to discover important environmental niches, are appropriate in number and generality to cover those niches, and can assume changing roles in response to a dynamic fitness landscape.

- We have applied the coevolutionary model to problems from the domains of concept learning and neural-network construction that can only be decomposed into subtasks with complex and difficult to understand interdependencies, and have compared and contrasted the resulting problem decompositions with those produced by task-specific non-evolutionary methods.

1.7 Dissertation Outline

In chapter 2 we begin with an introduction to evolutionary computation. This is followed by a discussion of several important issues related to the application of evolutionary algorithms to decomposable problems and a survey of previous related work. In chapter 3 we describe our computational model of cooperative coevolution, explain how the model addresses the issues raised in chapter 2, and discuss some of the advantages of the model over alternative approaches. The chapter concludes with a simple example of applying the model to a string covering problem. In chapter 4, a sensitivity analysis is performed on a number of characteristics of decomposable problems likely to affect the performance of the coevolutionary model. This analysis takes the form of focused experimentation using tunable test functions chosen specifically to measure the effect of these characteristics. The emergent problem decomposition properties of cooperative coevolution are studied in chapter 5 through a series of experiments involving the string covering problem from chapter 3. This is followed in chapter 6 by two case studies in the domains of artificial neural networks and concept learning that further explore the emergent decomposition capabilities of the model on problems that can only be decomposed into subtasks with complex and difficult to understand interdependencies. Finally, chapter 7 summarizes the results obtained in the dissertation and suggests some directions for future research.

Chapter 2

BACKGROUND AND RELATED WORK

A brief introduction to evolutionary computation, a discussion of the major issues related to the application of these algorithms to problems whose solutions require interacting coadapted subcomponents, and an overview of previous work addressing these issues is provided in this chapter. The chapter concludes with a summary of the limitations of previous approaches.

2.1 Evolutionary Computation

There are a number of different classes of algorithms that make up the field of evolutionary computation. In the early 1960's in Germany, Ingo Rechenberg, inspired by the "method of organic evolution", conceived the idea of solving optimization problems in aerodynamics by applying random mutations to vectors of real-valued shape defining parameters. This class of algorithms became known as *Evolution Strategies* (Rechenberg 1964). About the same time in the United States, work was being done independently by Lawrence Fogel et al. (1966) on the evolution of artificially intelligent automata represented as finite-state machines using a technique called *Evolutionary Programming*, and by John Holland (1975) on the analysis of a class of reproductive plans which were the precursors of what we now call *Genetic Algorithms*. More recently, a fourth class of evolutionary algorithms has emerged for generating Lisp programs. Early work in this area by Lynn Cramer (1985), Joe Hicklin (1986), and Cory Fujiki (1987) has been extended by John Koza (1989, 1992) and given the name *Genetic Programming*.

Although there are certainly differences among these four classes of algorithms, they are all based on the same fundamental principles of Darwinian evolution. These principles are as follows:

1. Organisms have a finite lifetime; therefore, propagation is necessary for the continuation of the species.
2. Offspring vary to some degree from their parents.
3. The organisms exist in an environment in which survival is a struggle and the variations among them will enable some to better adapt to this difficult environment.
4. Through natural selection, the better-adapted organisms will tend to live longer and produce more offspring.

5. Offspring are likely to inherit beneficial characteristics from their parents, enabling members of the species to become increasingly well adapted to their environment over time.

In summary, evolutionary computation is simply the application of these Darwinian principles to the solution of technical problems through computer simulation. We illustrate the evolutionary computation approach to problem solving with a simple example from the domain of function optimization.

Example 2.1 Given the following function:

$$f(\vec{x}) = \sum_{i=1}^{n} x_i^2,$$

our goal is to find values for the n independent variables such that the function is minimized. To achieve this goal through a computer simulation of evolution, we begin by selecting a representation for our population of competing solutions. One possibility is to use a binary representation for encoding elements of $\mathcal{R}^n$. In general, binary string representations are manipulated directly by genetic algorithms, real-valued vector representations are manipulated by evolution strategies, graph representations are manipulated by evolutionary programming, and tree representations are manipulated by genetic programming. We simulate propagation and inheritance through a process of recombination; that is, segments from one solution are combined with segments from another solution to create offspring. In this way, like begets like. The process of recombination, along with occasional random mutations, provide the source of variation. Death is simulated simply by replacing old solutions in the population with the new ones being created. Natural selection is accomplished by choosing the better problem solutions more often for recombination, thereby allowing them to pass their characteristics on to future generations more often than "less fit" solutions. Alternatively, less fit solutions could be chosen more often to be replaced. We determine the fitness of a solution by decoding it into the corresponding element of $\mathcal{R}^n$ and applying the resulting real-valued parameter vector to the target function $f(\vec{x})$. The smaller the function value produced, the higher the fitness of the solution. Given these conditions, Darwin's theory predicts that a population will adapt to its environment over time. In the context of our simple function optimization example, the theory predicts that solutions will evolve to produce results closer and closer to the desired minimum when applied to the target function.

2.1.1 Genetic Algorithms

Up to this point, we have intentionally kept our description of evolutionary computation at a high level of abstraction to emphasize the similarities between the various classes of evolutionary algorithms. As our primary focus in this dissertation is genetic algorithms, and to a lesser extent, evolution strategies, in this and the next section we provide a more detailed description of these two classes of algorithms. For a more comprehensive introduction to genetic algorithms than is provided here, including basic mathematical foundations, a detailed survey of applications, and a simple Pascal implementation, see (Goldberg 1989).

As the name implies, genetic algorithms model the evolutionary process at the level of the genome. Before a genetic algorithm can be used to solve a problem it is necessary to define a genetic code and a mapping between the genetic code and problem solutions. In biology, the genetic code of an organism is referred to as its *genotype*, and the instantiation of this code, that is, the physical realization of the being, is referred to as the organism's *phenotype*. Although we use this and other biological terminology here, we must emphasize that genetic algorithms are inspired by, rather than intended to be a true model of, evolutionary genetics. Therefore, terms such as genotype, phenotype, chromosome, and so on are used loosely.

In biological chromosomes, information is encoded within a strand of deoxyribonucleic acid (DNA) consisting of a long sequence of four bases: adenine, cytosine, guanine, and thymine. The entire genetic code of an organism is written in this four letter (A, C, G, and T) alphabet. In genetic algorithms, a chromosome is typically represented by a string written in a two-letter alphabet consisting of ones and zeros[1].

The process of designing a genetic code that can be used to construct problem solutions is illustrated with the following two examples from the domains of function optimization and artificial neural networks.

Example 2.2 A genetic algorithm is to be used to minimize a function $f(\vec{x})$ of integer-valued variables. Furthermore, we know that the function variables are constrained to the range $(0, 1023)$. A reasonable choice for this problem would be to use a binary coded decimal representation with ten bits allocated for each function variable. For example, given eight function variables, our chromosome would have a total length of 80 bits. The specific genotype-to-phenotype mapping would be as follows:

$$x_k = \sum_{i=1}^{10} 2^{i-1} chromosome[i + 10k],$$

where k is the index of a function variable. To compute the fitness of one of these individuals, we would construct an integer-valued parameter vector from its genetic code and apply the vector to the function we are minimizing. Smaller resulting function values will be produced by individuals with higher fitness.

Example 2.3 A genetic algorithm is to be used to determine the binary connection matrix of an artificial neural network. The network has 16 neural units—each with the potential of being connected to any of the other units. The connection matrix C is defined as follows:

$$c_{i,j} = \begin{cases} 1 & \text{if a connection exists between units } i \text{ and } j \\ 0 & \text{otherwise.} \end{cases}$$

A reasonable choice for this problem would simply be to linearize the 16×16 connection matrix into a binary chromosome of length 256. The specific genotype-to-phenotype mapping would be as follows:

$$c_{i,j} = chromosome[i + 16j].$$

[1] Although the most common representation used by genetic algorithms is a haploid chromosome implemented as a fixed-length binary string, these algorithms are not restricted to this representation.

$t = 0$
Initialize P_t to random individuals from $\{1, 0\}^l$
Evaluate fitness of individuals in P_t
WHILE termination condition is false **BEGIN**
    Select individuals for reproduction from P_t based on fitness
    Apply genetic operators to reproduction pool to produce offspring
    Evaluate fitness of offspring
    Replace members of P_t with offspring to produce P_{t+1}
    $t = t + 1$
    END

Figure 2.1: Canonical genetic algorithm

To compute the fitness of one of these individuals, we would build an artificial neural network using its genetic code as a connection matrix specification and train the network on some example problems. We would then use the ease of training, the accuracy of the final network output, or some combination of these two metrics as a fitness measure of the individual.

Once we have defined a genetic code, a mapping from the genetic code to an instantiation of a problem solution, and a method for evaluating the fitness of the solutions, the canonical genetic algorithm shown in figure 2.1 can be used to evolve a population of solutions. In the figure, P is a population, t is a discrete unit of time, and l is the chromosome length. The algorithm begins by initializing a population of individuals (genotypes). A random initialization would normally be used; however, for some applications knowledge may be available to enable the population to be more intelligently initialized. Each genotype is then decoded into a problem solution instantiation (phenotype) and its fitness evaluated. If a satisfactory solution does not exist in the initial population, individuals are chosen non-deterministically, based on their fitness, to reproduce. Once a reproductive pool has been selected, recombination is applied to create offspring and the offspring are mutated. Next, the fitness of each offspring is evaluated. Finally, old population members are, with equal likelihood, randomly replaced with the offspring to produce a new population. This select, recombine, evaluate, and replace cycle continues until a satisfactory solution is found or until a prespecified amount of time has elapsed.

As the algorithm runs, fitness-based selection allocates an increasing number of trials to regions of the solution space with an above average observed fitness. Genetic operators enable the algorithm to explore regions of the solution space not represented in the current population. This combination of exploitation and exploration enables the population to evolve to higher levels of fitness.

A commonly used selection technique called *fitness proportionate selection* is defined as

$$P_i(t+1) = \frac{f_i(t)}{\frac{1}{n}\sum_{j=1}^{n} f_j(t)}, \tag{2.1}$$

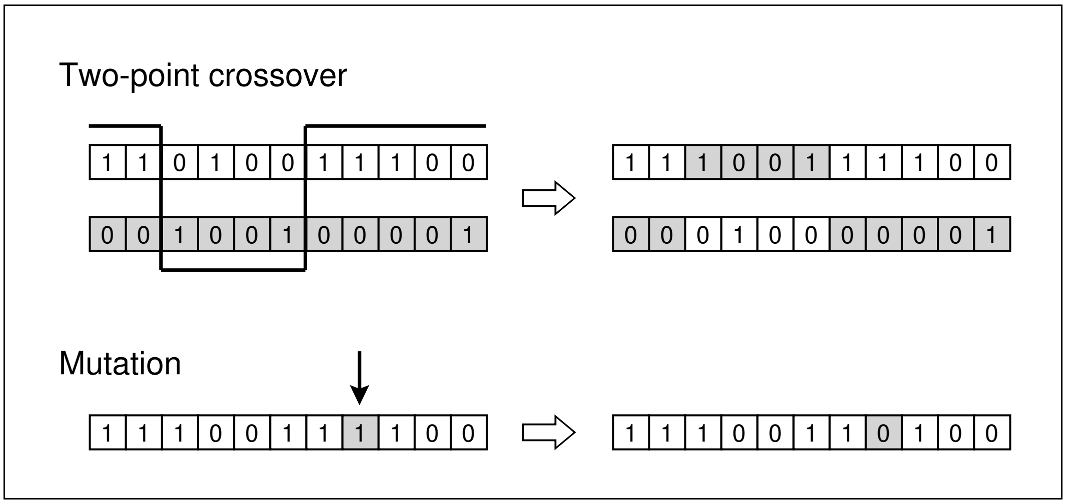


Figure 2.2: Two-point crossover and mutation operators

where n is the population size, P_i represents the selection probability of chromosome i, and f_i represents the fitness of i. The denominator of equation 2.1 on the preceding page computes the average fitness of the population at time t, where time is expressed discretely in generations. This selection technique will allocate proportionately more population slots to above average individuals and proportionately fewer slots to below average individuals.

The two most commonly used genetic operators, mutation and crossover, are shown in figure 2.2. The crossover operator will recombine genetic material from two parents to produce offspring. The version shown is called *two-point crossover* because it cuts the parent chromosomes at two random loci and swaps the segments between them. Another commonly used crossover operator called *uniform crossover* swaps each bit from one parent chromosome with the corresponding bit from another parent chromosome with a probability of 0.5. The mutation operator shown in the figure randomly flips bits in a chromosome to their opposite state. In genetic algorithms, mutation is normally used as a background operator, and as such, it is applied at a low rate. A typical mutation rate is $1/l$, where l is the number of bits in the chromosome. This results in an average of one bit mutated per chromosome.

One important distinction between mutation and crossover is that mutation has the ability to introduce new alleles into the genetic pool. For example, if every chromosome in the population contained a zero at locus seven, mutation could create an individual with a one at this position. Crossover does not have this characteristic. Specifically, it only recombines existing genetic material.

Here we have described the most commonly used genetic operators. Many other operators have been implemented, but they are used mostly in cases where the genetic algorithm utilizes some representation other than a binary string.

2.1.2 Evolution Strategies

Although most of our work has been in the area of genetic algorithms, we have not completely restricted ourselves to this class of evolutionary algorithms. Evolution strategies, for example, can also be enhanced by our macroevolutionary model of cooperative coevolution; and later in this dissertation we will explore the application of a “coevolution strategy” to the problem of artificial neural network construction. In this section, we briefly introduce some of the major forms of evolution strategies to provide the necessary background for this later study. For a more thorough description of evolution strategies than we provide here, including a mathematical analysis of the speed of convergence, a detailed comparison with other forms of numerical optimization, and Fortran implementations of a number of different variations, see (Schwefel 1995).

While genetic algorithms simulate evolution at the level of the genome, evolution strategies, and the other classes of evolutionary algorithms in use today, directly evolve phenotypes. Because evolution strategies were developed specifically for numerical optimization, they represent phenotypes as real-valued vectors.

The original evolution strategy was a two-membered scheme consisting of one parent and one offspring (Rechenberg 1964). In the basic algorithm, a parent is mutated to create an offspring, and the more highly fit of the two individuals survives into the next generation. This algorithm was later generalized into two multimembered forms—the so-called $(\mu{+}\lambda)$ and (μ, λ) evolution strategies. The parameters μ and λ refer to the number of parents and offspring respectively. In the $(\mu{+}\lambda)$ form, the parents and offspring are combined into a single selection pool and the μ best individuals survive into the next generation. In contrast, the (μ, λ) evolution strategy selects the μ survivors only from the set of offspring. As a result, no parents survive from one generation into the next. Given these definitions, the original two-membered scheme can be denoted by (1+1).

The ratio λ/μ of offspring to parents is usually seven or more. The larger this ratio, the greater the chance that each parent will produce at least one offspring superior to itself. Therefore, the difference between the $(\mu{+}\lambda)$ and (μ, λ) evolution strategies becomes less relevant when the offspring to parent ratio is large.

The canonical (μ, λ) evolution strategy is shown in figure 2.3 on the next page. The other forms are similar. A comparison with the canonical genetic algorithm shown in figure 2.1 on page 10 reveals what appears to be a major difference—while genetic algorithms generally select individuals to reproduce based proportionately on fitness and replace members from the previous population uniformly, the evolution strategy does just the opposite. That is, it selects individuals uniformly for reproduction and bases survival on fitness. In actuality, these approaches are but two sides of the same coin and have been shown experimentally to be equivalent (Bäck and Schwefel 1993).

Mutation, which is often the only evolutionary operator used, consists of perturbing each element of the connection-weight vector by an amount produced from a Gaussian distribution whose variance is adapted over time. Given the (1+1) strategy, the so-called 1/5 *success rule* is used to adjust the variance. Schwefel (1995) describes this rule-of-thumb as follows:

> From time to time during the optimum search obtain the frequency of successes, i.e., the ratio of the number of successes to the total number of trials (mutations). If the ratio is greater than 1/5, increase the variance, if it is less than 1/5, decrease the variance.

The 1/5 success rule was developed by Rechenberg (1973) as a result of his theoretical

$t = 0$
Initialize P_t to μ random individuals from $\mathcal{R}^n$
Evaluate fitness of individuals in P_t
WHILE termination condition is false **BEGIN**
  Clone individuals with equal likelihood from P_t to produce λ offspring
  Mutate offspring to create variation
  Evaluate fitness of offspring
  Select best μ offspring based on fitness to produce P_{t+1}
  $t = t + 1$
  END

Figure 2.3: Canonical (μ, λ) evolution strategy

investigation of the (1+1) strategy applied to two objective functions—the sphere and corridor models. The rule has since been shown to produce a high rate of convergence for these and other objective functions.

When using a multimembered strategy for evolving a population of μ parents and λ offspring, each individual consists of two real-valued vectors. One vector contains variable values and the other contains the corresponding standard deviations used by the mutation operator. Rather than using the 1/5 success rule, the multimembered strategies mutate the standard-deviation vectors each generation using a log-normal distribution as follows:

$$\vec{\sigma}_{t+1} = \vec{\sigma}_t e^{Gauss(0,\sigma')}, \tag{2.2}$$

where t denotes time expressed discretely in generations. The rate of convergence of the evolution strategy is sensitive to the choices of σ' and the initial setting of the standard-deviation vectors $\vec{\sigma}$. Unfortunately, no method for setting these values independent of the objective function is known. One recommendation by Schwefel (1995) is to set σ' as follows:

$$\sigma' = \frac{C}{\sqrt{|\vec{\sigma}|}}, \tag{2.3}$$

where C depends on μ and λ. He recommends that C be set to 1.0 given a $(10, 100)$ evolution strategy. Schwefel also recommends initializing $\vec{\sigma}$ using the equation

$$\sigma_k = \frac{R_k}{\sqrt{|\vec{\sigma}|}} \qquad \text{for } k = 1, 2, \ldots, |\vec{\sigma}|, \tag{2.4}$$

where the constant R_k is the maximum uncertainty range of the corresponding variable.

2.1.3 Evolutionary Algorithm Differences

One of the primary differences between genetic algorithms and the other evolutionary algorithms in use today is that genetic algorithms simulate evolution at the level of the genome, while the other evolutionary algorithms directly evolve phenotypes. As a result, the various algorithms use different representations for the population of evolving individuals. While genetic algorithms commonly use binary strings to represent individuals, evolution strategies most commonly use real-valued vector representations, and genetic programming uses tree representations. Evolutionary programming originally used graph representations, but now uses whatever phenotypic representation is appropriate for the problem being solved.

Another difference between the various evolutionary algorithms is in the genetic operators used. In contrast to the bit flipping mutation operator commonly used in genetic algorithms, most evolution strategies, for example, mutate genes through the addition of Gaussian noise. Evolutionary programming applied to finite state machines mutates its individuals by adding and deleting states, changing state transitions, and so forth. In addition, genetic algorithms and genetic programming emphasize the crossover operator, while both evolution strategies and evolutionary programming emphasize mutation.

Finally, there are differences in the basic flow of the various evolutionary algorithms. Genetic algorithms and genetic programming select individuals to reproduce based proportionately on fitness and replace members from the previous population uniformly. Evolution strategies and evolutionary programming assume the opposite strategy; that is, they select individuals for reproduction uniformly and base survival on fitness. As previously mentioned, these differences have little effect on the outcome of evolution. The important point is that all of these algorithms are based on the fundamental Darwinian principle of natural selection.

The advantages of one evolutionary algorithm over another are a matter of current debate. However, the macroevolutionary model we explore in this dissertation is not specific to any one evolutionary algorithm; therefore, this debate has little relevance here and will not be further addressed. For more information on this topic see, for example, a comparison of evolution strategies, evolutionary programming, and genetic algorithms within the context of the domain of function optimization by Bäck and Schwefel (1993).

2.2 Issues in Evolving Coadapted Subcomponents

Issues that must be addressed if we are to extend the basic computational model of evolution to provide reasonable opportunities for the emergence of coadapted subcomponents will now be examined in more detail. The issues include determining the precise characteristics of the problem decomposition, handling interdependencies between subcomponents, assigning appropriate credit to each subcomponent for its contribution to the problem-solving effort, and maintaining diversity in the environment. We will also discuss the inherent parallelism in evolution and additional opportunities that exist for parallel coevolutionary models.

2.2.1 Problem Decomposition

One of the primary issues that must be addressed if a complex problem is to be solved through the evolution of coadapted subcomponents is how to determine an appropriate number of subcomponents and the precise role each will play. We refer to this as *problem decomposition.* This is true whether the decomposition is at the macroscopic level, as in taking a divide-and-conquer approach in which a complex problem is broken into subtasks that are individually easier to solve, or at the microscopic level, in which a large collection of simple subcomponents form a composite solution.

An example of a macroscopic decomposition is solving an optimization problem of K independent variables using the *relaxation method*[2] (Southwell 1946; Friedman and Savage 1947). In the relaxation method, we cycle through the optimization of each of the K variables while holding the remaining $K-1$ variables fixed, thereby converting a single complex optimization task into K significantly easier subtasks. The microscopic decomposition is exemplified by rule based systems and artificial neural networks. In these systems each rule or neuron "fires" in response to only a subset of the circumstances it is exposed to. However, the collective behavior of the set of rules or neurons is expected to cover the entire space of possible circumstances. As a result, these are often called *covering problems* due to their similarity to the classic set covering problem from mathematics. More formally, let S_i represent the subset of circumstances the ith subcomponent correctly responds to. In a covering problem, one must construct a society of K subcomponents such that

$$M = \bigcup_{i=1}^{K} S_i, \tag{2.5}$$

where M represents the entire set of circumstances the system is expected to manage.

For some problems an appropriate decomposition may be known *a priori*. For example, given the problem of optimizing a function of K independent variables, it is reasonable in some cases to take a divide-and-conquer approach and decompose the problem into K subtasks as described above. However, there are many problems for which we have little or no information concerning the number or role of subcomponents that ideally should be in the decomposition. If the task is to learn classification rules, for example, we probably will not know beforehand how many rules will be required to cover effectively a given set of positive and negative examples or what characteristics of the examples each rule should respond to. Even decomposing the function optimization problem of K variables becomes non-trivial when there are nonlinear interactions between the variables. Perhaps rather than decomposing the problem into K single-variable optimization subtasks as described above, a better decomposition would be to cluster the optimization of interacting variables into common subtasks.

Given that an appropriate decomposition is not always obvious, it is extremely important that our problem-solving method addresses the decomposition task, either as an explicit component or as an emergent property of the method. Ideally when using an evolutionary algorithm, an appropriate decomposition will emerge as a result of evolutionary pressure; meaning, good decompositions will have a selective advantage over poor decompositions. For this to occur without human involvement, the basic evolutionary computation model needs to be extended both computationally and representationally.

2.2.2 Interdependencies Between Subcomponents

If a problem can be decomposed into independent subcomponents, each can be solved without regard to the others. Graphically, one can imagine each subcomponent evolving to

[2] This optimization method is known by a variety of names. Two of the more common alternatives are the *sectioning method* and the *coordinate strategy*.

achieve a higher position on its own separate *fitness landscape*, where a fitness landscape is simply a distribution of fitness values over either the genotype or phenotype space[3]. Unfortunately, many problems can only be decomposed 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 subcomponents to which it is linked (Kauffman and Johnsen 1991). Because evolutionary algorithms are adaptive, they are generally well suited to problems with a single dynamic fitness landscape resulting, for example, from a non-stationary objective function. However, given multiple dynamic fitness landscapes resulting from interdependencies, the standard evolutionary computation paradigm must be extended to allow some form of interaction between the linked subcomponents so they can coadapt.

The effect of interdependencies between subcomponents is illustrated with the following example from the class of covering problems.

Example 2.4 An evolutionary algorithm is to be used to solve a simple binary string covering task. We are given the following set of strings:

$$(0000, 0001, 0011, 0100, 0101, 1000)$$

which we will refer to as the *target set*. Our goal is to find the best possible three-element set of perfectly or partially matching strings called the *match set*. In other words, we must evolve a composite solution consisting of three subcomponents, where the subcomponents in this case are match set elements. The match strength between two strings is computed by summing the number of bits in the same position with the same value. The fitness of each match set element is computed by determining the match strength between it and each element of the target set, and summing the strengths in the cases where it matches the target set element better than any other individual in the match set. The fitness of the match set as a whole is computed by averaging the fitness values of each of its elements. If two or more match set elements tie for the lead in matching a particular target set element, the winner will be determined randomly. Since each match set element only gets credit when it matches a target string better than—or at least as good as—the other two coevolving elements, the three elements are interdependent.

Now assume that our evolutionary algorithm is run for some number of generations, and the match set and associated strengths illustrated in the left half of figure 2.4 on the facing page is generated. The solid connections between match set and target set elements indicate clear wins, while dashed connections indicate wins from randomly broken ties. The connections are labeled with match strengths. Using the fitness computation specified above, the first match set element, 0100, will have a fitness of 7; the second match set element, 0001, will have a fitness of 10; and the third match set element, 1000, will have a fitness of 4. By averaging the fitness of the three subcomponents, the match set is given a fitness of 7. Now assume the modification $1000 \rightarrow 0000$ is made to the third match set element, which results in the match set and strengths illustrated in the right half of the figure. The fitness of the third match set element will now be increased to 7 while the fitness of the first match set element is reduced to a value of 4—even though the element itself

[3]We will generally not be concerned with the additional complexity of operator neighborhoods in the fitness landscapes as studied by Jones (1995).

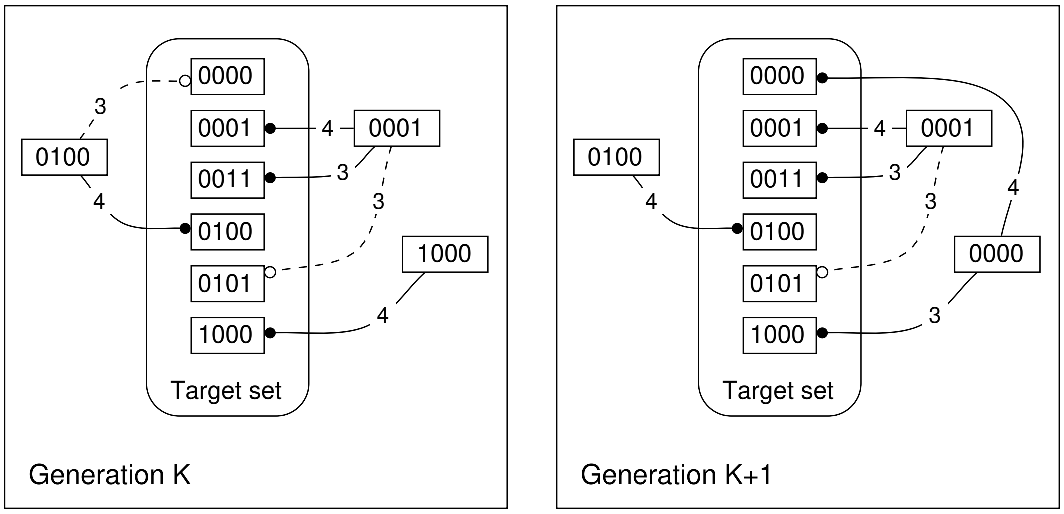


Figure 2.4: Match set, target set, and connection strengths before and after modification to a match set element

did not change. The fitness of the second match set element and the fitness of the match set as a whole remains unchanged. Using our earlier characterization, we would say that the fitness landscape associated with the first element has been warped by a change in the third element.

2.2.3 Credit Assignment

When a decomposable task is being covered collectively by a set of partial solutions, the determination of the contribution each partial solution is making is called the *credit assignment problem*. If we are given a set of rules for playing the game of chess, for example, it is possible to 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 less 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 credit assignment problem can be traced back to early attempts to apply machine learning to playing the game of checkers by Arthur Samuel (1959). The problem faced by Samuel was in determining how much credit or blame to assign to the elements of an attribute vector for correctly or incorrectly, assessing the worth of various checkerboard configurations.

One of the fundamental principles of Darwinian evolution is that the likelihood of an individual successfully passing its characteristics on to future generations is based on the fitness of the individual. If our goal is to use a computational model of evolution to solve a decomposable problem by way of a collection of coadapted subcomponents, there must be a process by which credit or blame is assigned to each of the subcomponents for their role in the health of the ecosystem. That is, we need to evaluate the subcomponents based on their contribution to the problem solving effort as a whole.

2.2.4 Population Diversity

If one is using an evolutionary algorithm to find a single individual representing a satisfactory solution to a problem, diversity only needs to be maintained in the population long enough to perform a reasonable exploration of the search space. As long as a good solution is found, it does not matter whether the final population consists of a single instance of this individual or has converged to a collection of clones of the individual. In contrast, solving a problem by way of a collection of coadapted subcomponents is not possible unless diversity is maintained to the end.

There is continuous pressure in an evolutionary algorithm driving the population to convergence. If one ignores for a moment stochastic effects and the disruptive effect of crossover and mutation, the canonical genetic algorithm shown in figure 2.1 on page 10 will allocate an increasing number of trials to above average regions of the sampled solution space and a decreasing number of trials to below average regions. This is a result of the Darwinian principal of natural selection, in which the more highly fit individuals produce a greater number of offspring than the less fit individuals. In general, this also holds true for evolution strategies, evolutionary programming, and genetic programming. With each new generation, the average fitness of the population will rise, making above average individuals increasingly exclusive. This, along with the effects of *genetic drift*, will eventually lead to a population consisting mostly of clones of a single highly fit individual.

Although the addition of stochastic effects and evolutionary operators make the precise trajectory of an evolutionary algorithm extremely difficult to predict, the pressure on the population to converge remains. Fortunately, there are known techniques for maintaining diversity within a single population, for example, the crowding and fitness sharing algorithms described in the section on multimodal function optimization beginning on page 22. Alternatively, we can achieve diversity in the ecosystem through genetically isolated species—the approach taken both by natural systems and by the model of cooperative coevolution explored in this dissertation.

2.2.5 Parallelism

Although parallel computing in the sense of utilizing multiple processors is not necessary for the evolution of coadapted subcomponents, it becomes a critical issue as we apply our algorithms to the solution of increasingly difficult problems. Because evolutionary algorithms evolve a population of solutions rather than a single solution, they are inherently parallel. For example, the fitness evaluations of the individuals can be done in parallel on separate processors. If determining the fitness of an individual is computationally expensive relative to the rest of the evolutionary algorithm, this trivial parallelism will result in a near linear speedup. Other ways of implementing a parallel evolutionary algorithm are also possible. They include the coarse-grain approach of evolving large subpopulations independently on a few processors and occasionally migrating individuals between processors, and the fine-grain approach of distributing individuals, or small subpopulations, among many processors and allowing them to interact with one another using localized mating rules.

Evolving a solution to a problem by way of a collection of coadapted subcomponents presents an opportunity for additional parallelism. Using a coarse-grain approach we can evolve each subcomponent in parallel on a separate processor. If the subcomponents are independent, no interprocessor communication is required until the final solution is assembled. Even with interdependencies between subcomponents, only occasional communication is required between processors. Due to this low communication overhead, near linear speedup will result independently of the computational expense of the fitness function.

2.3 Related Work

Previous examples of extending the basic evolutionary model to allow coadapted subcomponents can be divided into approaches that have restricted the ecosystem to a single population of interbreeding individuals and those whose ecosystem has consisted of multiple interacting populations.

2.3.1 Single Population Approaches

Classifier Systems

One of the earliest single-population methods for extending the basic evolutionary model to allow coadapted subcomponents to emerge is the *classifier system* (Holland and Reitman 1978; Holland 1986). Briefly, a classifier system is a rule-based system in which a population of stimulus-response rules is evolved using a genetic algorithm. Each rule is represented by a fixed-length ternary string consisting of the symbols 0, 1, and #. Each rule also has an associated strength. The operation of the classifier system consists of two phases. In the first phase, the population of classification rules is applied to some problem. Generally, a number of stimulus-response cycles will be executed in this phase. In the second phase, the genetic algorithm generates a new population of classification rules by selecting rules to reproduce based on the associated strengths, and applying genetic operators such as crossover and mutation to the selected rules. These two phases will alternate until the population of rules as a whole performs sufficiently well on the given task.

Along with the rules, there is a limited memory called the *message list*, and a matching function. Fixed-length binary messages are posted on the message list, either from an external environment or from the consequent of rules that have been activated. A rule is eligible to become active when its antecedent matches a message on the message list. The '#' symbol is used as a "don't care" in this matching process. The rules are managed through a simulated micro-economy. Specifically, all eligible rules participate in a bidding process in which only the rules making the highest bids are activated. The bid made by a rule is a function of its current strength and specificity. The dynamics of the micro-economy model are such that clusters of coadapted rules evolve over time, resulting in an emergent problem decomposition.

Computing the strength of each rule is the classic credit assignment problem described in section 2.2.3. The classifier system solves this problem using an algorithm called the *bucket brigade*. If the bucket brigade selects rule i for activation at time t, the strength of the rule is reduced according to the equation

$$\mathrm{strength}(i, t+1) = \mathrm{strength}(i, t) - \mathrm{bid}(i, t). \tag{2.6}$$

Simultaneously, all the rules j posting messages that are matched by i have their strengths increased according to the equation

$$\text{strength}(j, t+1) = \text{strength}(j, t) + \frac{\text{bid}(i, t)}{n}, \tag{2.7}$$

where n is the number of rules posting messages matched by i. Occasionally, there will be a positive environmental change and each currently active rule will have a payoff added to its strength.

Although the classifier system addresses the issues of emergent problem decomposition, interdependencies between subcomponents, credit assignment, and population diversity; it accomplishes this through an economic rather than a biological model and is specific to the task of evolving rule-based systems. In addition, the approach utilizes a number of centralized control structures that limit parallelism.

Other Approaches to Evolving Rules

An alternative approach to evolving stimulus-response rules with a genetic algorithm was developed by Smith (1983) in the system LS-1. Rather than evolving a population of rules as in classifier systems, each variable-length LS-1 chromosome represents an entire rule set. For historical reasons, representing an entire rule set with a chromosome is sometimes referred to as the *Pitt Approach*, while representing a single rule with a chromosome is referred to as the *Michigan Approach* (De Jong 1990). Since chromosomes are applied to a task individually rather than collectively when using the Pitt Approach, there is no longer an explicit credit assignment problem. However, the credit assignment problem still exists internal to the chromosome and surfaces in another form called *hitchhiking* (Das and Whitley 1991). Hitchhiking refers to a bad allele—a stimulus-response rule in this case—receiving a selective advantage over a good allele simply because the rest of the chromosome the bad allele appears in is very good. A disadvantage of LS-1 is that it is less modular than the classifier system and does not benefit from the mutual constraint that occurs when a group of individuals collectively solves a problem. The work of Smith has been greatly extended, combined with ideas from classifier systems, and given a richer rule representation in a system called SAMUEL (Grefenstette 1989; Grefenstette, Ramsey, and Schultz 1990).

Evolutionary computation has also been applied to the supervised learning of classification rules. Early work in this area includes a system developed by Janikow (1991, 1993) called GIL. This system uses a conjunctive representation for both the set of input examples and the evolved concept descriptors. Each conjunct is a tuple consisting of an attribute, a relation, and a set of values. A conjunction of these tuples is equivalent to a rule antecedent. The rule consequent is an implicit identification of the concept being learned. Each individual in the population consists of a disjunction of these conjunctions, equivalent to a rule set. Internally, the descriptors are mapped into a binary string chromosome for manipulation by a genetic algorithm. A similar approach has been taken in the design of a system called GABIL (De Jong, Spears, and Gordon 1993). Although the specific representation and genetic operators used by these two systems differ, they both adopt the Pitt Approach.

Another method for evolving classification rules with a genetic algorithm has been developed by Giordana et al. (1994) in a system called REGAL. In REGAL, the Michigan

Approach is taken in which each individual in the population represents a single rule; specifically, a conjunctive description in first order logic. A selection operator called *universal suffrage* clusters individuals based on their coverage of a randomly chosen subset $\mathcal{E}$ of the positive examples. If no individuals from the current population cover a particular element of $\mathcal{E}$, a covering individual is created with a seeding operator. By combining a single individual from each cluster, a disjunctive description guaranteed to cover $\mathcal{E}$ is formed. 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. Universal suffrage addresses the issues of problem decomposition and credit assignment, but is specific to the task of concept learning. We will later discuss a distributed version of REGAL in section 2.3.2

Evolving Artificial Neural Networks

The evolution of artificial neural networks is another example of a covering problem, and as such, it is similar to the problem of evolving rule-based systems. Therefore, it is not surprising that the approaches for evolving solutions to these two types of problems are also similar. Although there have been a number of studies proposing mappings between the neural network model and classification systems (Compiani, Montanari, Serra, and Valastro 1988; Belew 1989; Farmer 1991), most of the early examples of evolving artificial neural networks have used a technique in which all the neurons in the network are represented with a single chromosome. Since this is analogous to the Pitt Approach to learning classification rules, we will continue to use this terminology in the context of neural network evolution.

One of the first successful attempts to use a genetic algorithm to evolve neural networks using the Pitt Approach was reported by Montana and Davis (1989). More specifically, they evolved just the connection weights for a feed-forward network having a fixed topology. The genetic algorithm was intended to replace the back-propagation algorithm (Rumelhart, Hinton, and Williams 1986)—a gradient-descent approach to learning multilayered feed-forward network connection weights that was the best technique known at the time. However, it was later discovered that incompatible representations for equivalent networks coexisted in the population and produced inferior offspring when mated, reducing the effectiveness of the genetic algorithm substantially (Whitley, Starkweather, and Bogart 1990). This is called the *competing conventions* problem.

An early Pitt Approach to using a genetic algorithm to evolve network topology was explored by Miller et al. (1989). In their system, a chromosome represented the entire topology of the network by linearizing the binary connection matrix of all the nodes as we described in example 2.3 on page 9. To evaluate an individual, a network was constructed with the connectivity specified by its chromosome and trained with the back-propagation algorithm for a fixed number of epochs. The fitness of the individual was taken to be the sum-squared error of the network after the final training epoch.

More recently, the Pitt Approach has been used to evolve both connection weights and topology. For example, emergent approaches were taken independently by Potter (1992) and Karunanithi et al. (1992) in which the topology evolved as a result of the connection weight learning process. They accomplished this using a cascade architecture developed by Fahlman (1990) in which new neural units are added when learning approaches an asymptote. An alternative approach, in which both the topology and connection weights are explicitly represented by an individual, was taken by Spofford and Hintz (1991).

In contrast to these earlier systems for evolving artificial neural networks, Moriarty (1996) has developed a coevolutionary model more similar to the Michigan Approach in which each individual represents a single neuron. The system is evolved with a genetic algorithm and is called SANE (Symbiotic Adaptive Neuro-Evolution). In SANE, the genotype of each neuron specifies which input and output nodes it connects to and the weights on each of its connections. A single generation consists of many cycles of selecting a random subset of neurons from the population, connecting them into a functional neural network, evaluating the network, and passing the resulting fitness back to each of the participating neurons. Each neuron in the collaboration sums this fitness with the fitness values it has received from earlier collaborations. By allowing a neuron to participate in a number of different networks, its average fitness becomes a measure of how well it collaborates with other neurons in the population to solve the target problem. Rewarding individuals based on how well they collaborate results in the long term maintenance of population diversity and a form of emergent decomposition. Currently, SANE is limited to the evolution of artificial neural networks. Although the system could probably be adapted to other types of covering problems, its use of a single interbreeding population confines its application to problems in which all subcomponents share a common representation.

Finally, techniques such as *shaping* or *chaining* have been used to train animals to perform complex tasks in stages by breaking the tasks down into simpler behaviors that can be learned more easily, and then using these simpler behaviors as building blocks to achieve more complex behavior (Skinner 1938). A similar approach has been taken by de Garis (1990) to evolve artificial neural networks for controlling simulated creatures, for example, an artificial lizard called LIZZY. Rather than evolving a single large neural network to control the creature, de Garis first hand-decomposes the problem into a set of component behaviors and control inputs. A genetic algorithm is then used to evolve small specialized neural networks individually, which de Garis calls *GenNets*, that exhibit the appropriate behaviors. The resulting collection of GenNets implementing the various behavior and control functions are then "wired" together to form a completely functional creature. Shaping has also been 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.

Multimodal Function Optimization

Functions with more than one maximum or minimum are called *multimodal* functions. When solving these problems, we are often interested in finding all the significant peaks or valleys rather than just a single global optimum. The key issue here is the preservation of population diversity.

An early technique for preserving diversity called *crowding* was introduced by De Jong (1975). Crowding assumes a genetic algorithm model in which each iteration consists of creating a single offspring, evaluating its fitness, and inserting it back into the population; that is, during each generation a single individual is born and a single individual dies. This is referred to as a *steady-state* model. The crowding algorithm is applied during the replacement phase of the steady-state model by choosing a set of individuals randomly from the population, determining which individual in the set is the most similar to the offspring, and replacing this individual with the offspring. Crowding is able to preserve population diversity for a time; but eventually, evolutionary pressure and genetic drift will result in population convergence.

Another technique called *fitness sharing*, developed by Goldberg and Richardson (1987), maintains population diversity when evolving solutions to multimodal functions by modifying the fitness evaluation phase of the genetic algorithm. The following sharing function is defined:

$$\text{share}(d_{ij}) = \begin{cases} 1 & \text{if } d_{ij} = 0 \\ 1 - \left(\frac{d_{ij}}{\sigma_s}\right)^{\alpha} & \text{if } d_{ij} < \sigma_s \\ 0 & \text{otherwise,} \end{cases} \tag{2.8}$$

where d_{ij} represents the *distance* between individuals i and j, σ_s is a cluster radius, and α controls how quickly sharing drops off as distance increases. Distance can be measured in either phenotype or genotype space, although phenotype space generally gives the best results. The sharing function is used in the fitness computation as follows:

$$f_i' = \frac{f_i}{\sum_{j=1}^{n} \text{share}(d_{ij})}, \tag{2.9}$$

where f is the raw fitness and n is the population size. When fitness sharing is applied, the population evolves into a number of clusters of individuals about each maximum. Using biological terminology, the regions around the maximums are referred to as *niches*[4]. Furthermore, the number of individuals in each niche is proportional to the fitness of its associated maximum. The problems with fitness sharing are that it is expensive computationally because the distance between all pairs of individuals must be computed, and the technique is sensitive to the σ_s parameter. Although a method for setting σ_s has been developed, it makes strong assumptions about the shape of the multimodal function (Deb and Goldberg 1989).

An iterative approach to evolving solutions to multimodal functions called the *sequential niche technique* was developed by Beasley et al. (1993). This approach borrows from fitness sharing the idea of fitness devaluation based on a distance metric. However, rather than continuously modifying the fitness function based on the distance between individuals in the current population, the sequential niche technique reduces the fitness of an individual based on its distance from peaks found in previous runs of the genetic algorithm. If the location of five peaks were desired, for example, five complete runs of the genetic algorithm would be performed. Each run is terminated when improvement approaches an asymptote. Although this technique is not as computationally expensive as fitness sharing, it suffers from the same sensitivity to σ_s.

A different sort of technique explored by Perry (1984), and later by Spears (1994), for maintaining genetic diversity in multimodal function optimization is to use tag bits to group individuals into subpopulations. The basic idea is to reserve a region of the chromosome for use as a subpopulation label. If n chromosome bits are allocated for this purpose, each individual would be labeled as belonging to one of 2^n unique subpopulations. Mating is only allowed between individuals belonging to the same subpopulation. Since the tag bits

[4]Maintaining niche coverage implies more than diversity. For example, a randomly initialized population is diverse but probably will not be clustered into niches.

are mutated along with the rest of the chromosome, individuals can in effect move from one subpopulation to another. Spears normalized the fitness of each individual by dividing its raw fitness by its subpopulation size. The combination of tag bits and the normalized fitness metric enables individuals to be maintained on several peaks of a multimodal function. The number of peaks that can be covered is a function of the population size and the number of allocated tag bits. A disadvantage of the tag bit approach is that both of these parameters must be prespecified by the user.

Modeling the Immune System

A more recent approach to maintaining population diversity, which also partially addresses the issues of emergent problem decomposition and credit assignment, is to model a biological system that has evolved to satisfy the same requirements, specifically, the vertebrate immune system. The role of the immune system is to protect our bodies from infection by identifying and destroying foreign material. Immune system molecules called *antibodies* play an important role in this process by first identifying foreign molecules, and then tagging them for removal. Molecules capable of being recognized by antibodies are called *antigens*. For a more detailed description of the immune system, see section 6.2.1 beginning on page 100.

The interaction between antibodies and antigens was first modeled using a binary string representation by Farmer, Packard, and Perelson (1986). Their primary interest was in studying a number of theories concerning *idiotypic networks*, one of the hypothesized regulatory mechanisms of the immune system. The model was later simplified and a standard genetic algorithm used to evolve a population of antibody strings that covered a given set of antigen strings (Stadnyk 1987). In this work, the crowding strategy developed by De Jong (1975) was utilized to keep the population of antibodies from converging. More recently, this work has been further extended (Forrest and Perelson 1990; Smith, Forrest, and Perelson 1993; Forrest, Javornik, Smith, and Perelson 1993). Although this more recent work also evolves antibodies with a single-population genetic algorithm, it introduces a new stochastic algorithm called *emergent fitness sharing* that simulates the interactions between antibodies and antigens responsible for preventing a homogeneous population of antibodies from evolving within our bodies.

The algorithm for emergent fitness sharing is shown in figure 2.5 on the next page. Within the context of the canonical genetic algorithm, this procedure would be executed once each generation. A single execution computes the fitness of each of the antibodies in the population through an iterative process. During each iteration, a single antigen is selected randomly from a fixed collection of these foreign molecules, and an antibody set of size σ is chosen randomly without replacement from an evolving population. The chosen antibodies then hold a tournament to determine who matches the antigen most closely, and the winner receives a fitness increment based on the quality of the match. The fitness evaluation algorithm requires C iterations, where C is large enough to ensure that each antigen will be selected at least once. A large value for C also gives each antibody many opportunities to participate in antigen matching tournaments.

Although emergent fitness sharing enables the genetic algorithm to maintain diversity and evolve a mixture of generalists and specialists, the size, σ, of the antibody set chosen

```
i = 0
WHILE i < C BEGIN
    Randomly select single antigen j from fixed set of antigens
    Randomly select set of antibodies S of size σ from antibody population
    FOR each antibody k in S
        score_k = quality of match between j and k
    Choose best scoring antibody in S (randomly break ties)
    Increment fitness of best antibody by its score
    i = i + 1
    END
```

Figure 2.5: An algorithm for modeling emergent fitness sharing in the immune system

each iteration strongly influences diversity and emergent generalization. In some respects, the σ parameter is similar to the cluster radius parameter in the fitness sharing algorithm described earlier. A large σ encourages more population diversity and more specialization, while a small σ will result in less diversity and more generalization. No method of setting σ has been reported other than trial and error; that is, the human is very much involved in tuning the algorithm to produce good results.

As an aside, a slightly modified version of emergent fitness sharing has been used to evolve strategies for the two-player iterated prisoner's dilemma (Darwen and Yao 1996; Darwen 1996). In this game, two players simultaneously and independently decide to either cooperate with one another or defect. If both players cooperate, they each receive a larger reward than if they both defect; however, if one player defects and the other cooperates, the defector receives an even greater reward; see, for example, (Axelrod 1984). In the Darwen and Yao study, the motivation for using emergent fitness sharing is to maintain a diverse collection of strategies in the population to avoid over-specialization. Once the population is sufficiently evolved, a tournament is held to determine which strategies work best against each other. When confronted with a new opponent, the Hamming distance between it and the evolved strategies is used to classify the opponent and select a counter-strategy from the population.

Evolving Subroutines

Work on extending the basic evolutionary model to allow coadapted subcomponents is not limited to genetic algorithms. Within the domain of genetic programming, Koza (1993) has reported on the beneficial hand-decomposition of problems into a main program and a number of subroutines. Rosca and Ballard (1994, 1996) have taken a more emergent approach through the exploration of techniques for automatically identifying blocks of useful code, generalizing them, and adapting the genetic representation to use the blocks as subroutines in future generations. However, all genetic programming approaches to date have focused on the coadaptation of structure that is highly specific to the evolution of computer programs.

2.3.2 Multiple Population Approaches

The Island Model

Over sixty years ago, population geneticist Sewall Wright (1932) hypothesized that isolated subpopulations with occasional migration between one another would collectively maintain more diversity and reach higher fitness peaks than a single freely interbreeding population. This idea, which Wright referred to as the *island model*, was confirmed using a genetic algorithm by Grosso (1985). Grosso also found that the model was sensitive to the rate of migration—too frequent migration resulted in results similar to a single freely interbreeding population, and too little migration resulted in diversity but substandard fitness. These ideas were also explored by Cohoon et al. (1987), who investigated the theory of *punctuated equilibria* in which subpopulations pass through alternating phases of rapid evolution and stasis; Petty et al. (1987), who used frequent migration to enable small populations distributed over multiple processors to act as though they were a single large population; and Tannese (1987, 1989), who experimented systematically with different migration rates. In addition, Whitley and Starkweather (1990) applied a genetic algorithm with distributed subpopulations to a variety of problems, and were consistently able to optimize larger problems with less parameter tuning using this technique.

Although the island model improves the performance of evolutionary algorithms by maintaining more diversity in the ecosystem and providing more explicit parallelism, it does not address any of the other issues related to the evolution of coadapted subcomponents. Giordana et al. (1996) have taken a step in this direction by combining the island model with the universal suffrage operator in a distributed version of their REGAL system, described previously under single population approaches. In the distributed version of REGAL, each island consists of a population of conjunctive descriptions that evolve to classify a subset of the positive examples. The universal suffrage operator is applied within each island population to ensure that all its assigned examples are covered. Migration of individuals occurs between islands at the end of each generation. A "supervisor process" determines which examples are assigned to each island, and occasionally reassigns them to encourage a one-to-one correspondence between islands and modalities in the classification theory. In other words, each island will ultimately produce a single conjunctive description that, when disjunctively combined with a conjunctive description from each of the other islands, will correctly classify all the positive and negative examples. This approach achieves good problem decompositions but is highly task-specific.

Fine-Grain Approaches

Population diversity can also be maintained with a fine-grain parallel approach in which individuals, or small subpopulations, are distributed among many processors and allowed to interact with one another using localized mating rules (Mühlenbein 1989; Gorges-Schleuter 1989; Manderick and Spiessens 1989; Davidor 1991; Spiessens and Manderick 1991). With this technique, population diversity is a function of the communication topology of the processors. The greater the communication distance between two processors, the more likely it will be that their respective subpopulations will differ. However, diversity maintained by topology alone is transitory; that is, if the system is run for many generations, the population will converge (McInerney 1992). Furthermore, the fine-grain approach alone is not sufficient to allow the evolution of coadapted subcomponents.

Competitive Models

Biologists have theorized that one response of large multicellular organisms to the presence of pathogens such as parasites is an increase in genetic diversity (Hamilton 1982). Hillis (1991) has applied a model of hosts and parasites[5] to the evolution of sorting networks using a genetic algorithm. One species (the host population) represents sorting networks, and the other species (the parasite population) represents test cases in the form of sequences of numbers to be sorted. Hillis takes a fine-grain parallel approach in which individuals evolve on a two-dimensional toroidal grid of processing elements. The members of a species interbreed with one another using a Gaussian displacement rule, while interaction between parasites and hosts is limited to the pairs of individuals occupying the same grid location. The interaction between species takes the form of complementary fitness functions; that is, a sorting network is evaluated on how well it sorts the test cases that coexist at its grid location, while the test cases are evaluated on how poorly they are sorted. The host and parasite species are genetically isolated and only interact through their fitness functions. Because the host and parasite populations do not interbreed, they are full-fledged species in a biological sense.

A competitive model was also used by Rosin and Belew (1995) to solve a number of game learning problems, including tic-tac-toe, nim, and go. In their model, the two species represent opponents in the game. Rather than using a grid topology as in Hillis' work to determine the pattern of interaction between species, they used an algorithm called *shared sampling*. Briefly, in shared sampling each member of one species is matched against a sample of opponents from the previous generation of the other species. The sample of opponents is chosen based on their ability to win games. Furthermore, the sample is biased toward opponents who have the ability to defeat competitors that few others in their respective population can beat. The fitness evaluation procedure used by Rosen and Belew, which they call *competitive fitness sharing*, rewards an individual based on the number of opponents from the sample it defeats. The amount of reward resulting from each win is a function of the number of other individuals in the population who can defeat the same opponent. In some respects, this fitness evaluation procedure is similar to the algorithm for emergent fitness sharing described earlier under single population approaches.

Both of these competitive models have demonstrated that this form of interaction between species helps to preserve genetic diversity and results in better final solutions when compared with non-coevolutionary approaches. It can also result in a more computationally efficient fitness function; for example, regarding Hillis' work, each sorting network only needs to be evaluated on a small collection of coevolving test cases. Two limitations of these specific approaches to competitive coevolution are that they involve a hand-decomposition of the problem and they have a narrow range of applicability.

[5] Although it was called a host-parasite model by Hillis, technically his work is an example of a *competitive* model. In contrast, a true host-parasite model is exploitative (Smith 1989).

Cooperative Models

A genetic algorithm for coevolving multiple cooperative species was applied to job-shop scheduling by Husbands (1991). Typically, job-shop scheduling systems generate an optimum schedule given a set of descriptions for machining parts called *process plans*. In the Husbands system, each species represents alternatives for a single process plan, and is genetically isolated from the other species through the use of multiple populations. Husbands also evolved a species of individuals, called the *arbitrator* species, for resolving conflicts when two process plans need to use the same machine at the same time. In effect, the arbitrator performs the scheduling task. Husbands used a genetic representation and operators tailored to process plans. The fitness of each process plan was computed as the machining and setup cost, plus an additional cost if the machining of the part was delayed due to having to wait for a machine to become available. The fitness of the arbitrator was computed as a function of the total amount of time spent in the job shop waiting for machines to become available and the time required to finish machining all the parts. The pattern of interaction between species was determined by first ranking the individuals in each population based on fitness and then combining them with individuals having equal rank from the other species to form complete solutions. In other words, the best individuals from each species form a complete solution; the second best individuals form a complete solution; and so on. Since the required number of process plans and what they needed to accomplish were known beforehand, the problem was simply decomposed by hand.

Paredis (1995) used a cooperative, also referred to as *symbiotic*, two-species model for the solution of a deceptive problem introduced by Goldberg et al. (1989). One species represents problem solutions as 30-bit vectors, and the other species represents permutations on the order of the bits in the solutions. The motivation for this decomposition was that some permutations of the bits enable a genetic algorithm to solve this particular deceptive problem more easily. Since it was not known beforehand which permutations would be helpful, they were coevolved along with the solutions. Interaction between the two species consisted of grouping each individual from the solution population with two randomly chosen individuals from the permutation population and applying the two resulting combinations to the deceptive problem. This process was repeated a number of times and the fitness values averaged. As with the other coevolutionary models, this system involves a hand-decomposition of the problem.

2.4 Limitations of Previous Approaches

Although there have been many earlier approaches to extending the evolutionary computation model to allow the emergence of coadapted subcomponents, no single approach preserves the generality of the basic model while satisfactorily addressing the issues of problem decomposition, interdependencies between subcomponents, credit assignment, population diversity, and parallelism.

To summarize the limitations of these previous approaches, classifier systems address several of the issues we have outlined, but are limited to the task of evolving rule-based systems. They achieve this through centralized control structures that limit parallelism and an economic rather than biological model. The Pitt Approaches—both for evolving rule-based systems and artificial neural networks—avoid many of the issues we have discussed by representing complete solutions as individuals rather than as a collection of interacting subcomponents. However, they are not as modular as Michigan Approach systems such as classifier systems, and thus are limited by the scale-up problem. Both the single population and multiple population REGAL systems are limited to concept learning from preclassified examples. The SANE system addresses most of the issues but is currently limited to the evolution of artificial neural networks. The shaping technique and both the competitive and cooperative coevolutionary models involve a hand-decomposition of the problem. The island model and the approaches directed at multimodal function optimization only handle the population diversity issue; and, in the case of the island model, the issue of parallelism. The immune system model is sensitive to parameters that must be hand-tuned. The genetic programming approaches are specific to the evolution of computer programs. Finally, all the single gene-pool approaches, which include all but the coevolutionary models, are limited to the evolution of subcomponents that share a common representation.

Chapter 3

ARCHITECTURE

In the previous chapter we discussed a number of important issues that must be addressed if we are to extend the basic computational model of evolution to allow the emergence of coadapted subcomponents. The issues include problem decomposition, interdependencies between subcomponents, credit assignment, population diversity, and parallelism. We also described a number of previous evolutionary approaches that address these issues to a varying degree and pointed out some of their limitations. Our conclusion was that no single approach satisfactorily addresses all the issues while maintaining the generality of the basic evolutionary computation model.

In this chapter, we describe a macroevolutionary approach we call cooperative coevolution, which combines and extends ideas from these earlier evolutionary approaches to improve their generality and their ability to evolve interacting coadapted subcomponents without human involvement. The chapter is organized in four main sections. In the first section, we describe the basic cooperative coevolutionary model. Next, we discuss how it addresses the above-mentioned issues. We then discuss some additional advantages of the model such as speciation through genetic isolation, generality, and efficiency. The chapter concludes with a simple instantiation of the model in which it is applied to the string matching task described in example 2.4 beginning on page 16.

3.1 A Model of Cooperative Coevolution

In cooperative coevolution, we model an ecosystem consisting of two or more sympatric species having an ecological relationship of mutualism. The species are encouraged to cooperate with one another by rewarding them based on how well they work together to solve a target problem. As in nature, the species are genetically isolated. We enforce genetic isolation simply by evolving the species in separate populations. Although the species do not interbreed, they interact with one another within a domain model.

The canonical cooperative coevolutionary algorithm is shown in figure 3.1 on the next page. It begins by initializing a fixed number of populations—each representing a separate species. We will describe later how this algorithm can be extended to enable an appropriate number of species to emerge without being prespecified by the user. The fitness of each member of each species is then evaluated by forming collaborations with individuals from other species (see below). If a satisfactory solution to the target problem is not found initially, all the species are further evolved. For each species, this consists of selecting

$t = 0$
FOR each species S
  Initialize $P_t(S)$ to random individuals from $\{1, 0\}^l$
FOR each species S
  Evaluate fitness of individuals in $P_t(S)$
WHILE termination condition is false **BEGIN**
  FOR each species S **BEGIN**
    Select individuals for reproduction from $P_t(S)$ based on fitness
    Apply genetic operators to reproduction pool to produce offspring
    Evaluate fitness of offspring
    Replace members of $P_t(S)$ with offspring to produce $P_{t+1}(S)$
    END
  $t = t + 1$
  END

Figure 3.1: Canonical cooperative coevolution algorithm

individuals to reproduce based on their fitness, for example, with fitness proportionate selection; applying genetic operators such as crossover and mutation to create offspring; evaluating the fitness of the offspring; and replacing old population members with the new individuals. Although the particular algorithm shown is an extension of the canonical genetic algorithm, other evolutionary algorithms could be extended in a similar fashion.

A more detailed view of the fitness evaluation of individuals in one of the species is shown in figure 3.2 on the following page. Individuals are not evaluated in isolation. Instead, they are first combined in some domain-dependent way with a representative from each of the other species. We refer to this as a *collaboration* because the individuals will ultimately be judged on how well they work together to solve the target problem. There are many possible methods for choosing representatives with which to collaborate. In much of our work, the current best individual from each species is chosen as a representative; however, in some cases this strategy is too "greedy" and other strategies may be 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. The selection of a single, or at most a few, representatives from each species avoids an exponential increase in the number of collaborations that must be evaluated. The final step in evaluating an individual is to apply the collaboration to the target problem and estimate the fitness. This fitness is assigned strictly to the individual being evaluated and is not shared with the representatives from the other species that participated in the collaboration.

A graphic illustration of the interaction that occurs between species is shown in figure 3.3 on page 33. From this figure one can see that the representatives only provide context and do not receive any fitness evaluation. Although most of our implementations of this model have

Choose representatives from other species
FOR each individual i from S requiring evaluation **BEGIN**
Form collaboration between i and representatives from other species
Evaluate fitness of collaboration by applying it to target problem
Assign fitness of collaboration to i
END

Figure 3.2: Fitness evaluation of individuals from species S

utilized a synchronous pattern of interaction, this is certainly not necessary. That is, each species could be evolved asynchronously on its own processor as long as all the individuals belonging to a single species are evaluated within the same context. In addition, it is not necessary for the representatives to be updated each evolutionary cycle—further reducing the communication overhead of a parallel implementation of the model. The figure shows the interaction between individuals occurring within the context of a domain model and is nebulous about how the collaborations are actually constructed. We have intentionally been vague on this point because collaboration construction depends on what the species represent, which in turn depends on the problem domain. This can best be illustrated with a few simple examples.

Example 3.1 Cooperative coevolution is to be used to maximize a function $f(\vec{x})$ of n independent variables (Potter and De Jong 1994). The problem is hand-decomposed into n species—one species for each independent variable. In other words, each species represents alternative values for a particular variable. Interaction consists of selecting an individual (variable value) from each species and combining them into a vector that is applied to the target function. An individual is rewarded based on how well it maximizes the function within the context of the variable values selected from the other species.

Example 3.2 Cooperative coevolution is to be used to develop a rule-based system of behaviors for an autonomous robot (Potter, De Jong, and Grefenstette 1995). Each species represents alternative rule sets for the implementation of a particular behavior, perhaps evolved using a system such as SAMUEL described in chapter 2. Interaction consists of selecting an individual (rule set implementing a behavior) from each species and using them collectively to control the robot. We may also need to coevolve an arbitrator species to integrate the behaviors. An individual is rewarded based on how well it complements the behaviors selected from the other species to enable the robot to perform its function. We do not know *a priori* what behaviors will enable the robot to do its job most effectively, so we initialize a sufficient number of species randomly and let their specific roles emerge as a result of evolutionary pressure.

In the previous two examples we either knew exactly how many species were required or were able to place a reasonably small upper bound on the number. In other domains, such as evolving artificial neural networks, we may have little or no prior knowledge to help

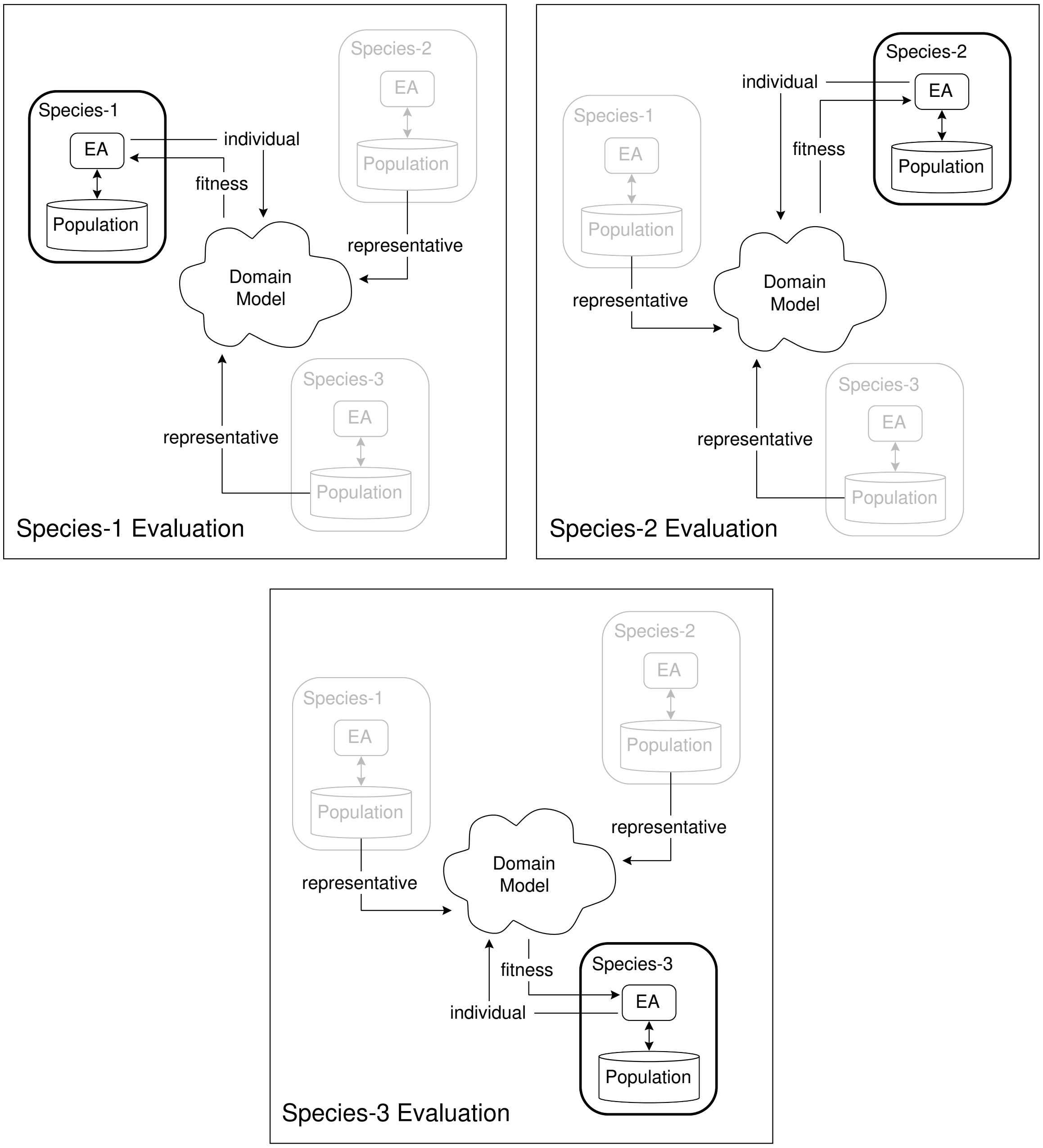


Figure 3.3: Model of species interaction

```
IF evolution has stagnated THEN BEGIN
    FOR each species S BEGIN
        Check contribution of P_t(S)
        Remove S from ecosystem if unproductive
        END
    Initialize P_t(S_new) to random individuals from {1,0}^l
    Evaluate fitness of individuals in P_t(S_new)
    END
```

Figure 3.4: Birth and death of species

us make this determination. Ideally, we would like an appropriate number of species to be an emergent property of cooperative coevolution. Figure 3.4 shows one possible algorithm for achieving this through a model of the birth and death of species. The model works as follows. If evolution stagnates, it may be that there are too few species in the ecosystem from which to construct a good solution; therefore, a new species will be created and its population randomly initialized. 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

$$f(t) - f(t-K) < C, \tag{3.1}$$

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

Example 3.3 Cooperative coevolution is to be used to grow and train a two-layer feed-forward neural network with x input units, y output units, and an unspecified number of hidden units (Potter and De Jong 1995). Each species represents alternatives for one of the hidden units in the network. Specifically, an individual consists of a set of genes that code which input and output nodes a hidden unit is connected to, the weights on its connections, and a threshold function. Interaction consists of selecting an individual (hidden unit specification) from each species and combining them into a functional network that is applied to a target problem. An individual is rewarded based on how well it functions with the hidden units from the other species as a complete network. Initially, the ecosystem contains a small number of species. Over time, the number of species will increase until a network of sufficient size and complexity to handle the target problem can be constructed. In this case both the number of hidden nodes and their function emerges as a result of evolutionary pressure.

Before we move on to a discussion of how this model addresses the important issues in evolving coadapted subcomponents introduced in chapter 2, one more point needs to be made concerning terminology. In the traditional evolutionary computation model, a *generation* is defined to be one pass through the select, recombine, evaluate, and replace cycle shown in figure 2.1 on page 10. In biological terms, this is roughly equivalent to the average time span between the birth of parents and their offspring. The length of this time span depends on the species in question. In our computer model, the generational time span could be measured in fitness evaluations, CPU cycles, or simply in "wall clock time". This creates a conceptual problem when we move from the traditional single population model to the multiple population model of coevolution shown in figure 3.1 on page 31. If we simply expanded the term generation to mean one pass through the select, recombine, evaluate, and replace cycles of all the species being coevolved, the generational time span as computed with any of the three metrics just mentioned would depend on the number of species in the ecosystem. However, in nature the number of species in an ecosystem has little to no effect on the time span between the birth of parents and their offspring.

To resolve this conflict, throughout this dissertation a distinction will be made between *generations* and *ecosystem generations*. We define a generation to be a complete pass through the select, recombine, evaluate, and replace cycle of a single species while an ecosystem generation is defined to be an evolutionary cycle through all species being coevolved. This terminology is consistent with that previously used by Jones (1995). In general, an ecosystem generation will consist of n times more fitness evaluations than a generation, where n is the number of currently existing species. Given these definitions, the index t shown in figure 3.1 counts the number of ecosystem generations that have occurred.

3.2 Issues Revisited

In chapter 2 we discussed a number of important issues that must be addressed if we are to evolve coadapted subcomponents. These issues include problem decomposition, interdependencies between subcomponents, credit assignment, and maintaining diversity in the environment. We also pointed out that parallelism becomes a critical issue as we try to solve increasingly difficult problems. We now revisit these issues and describe how they are addressed by our model of cooperative coevolution.

3.2.1 Problem Decomposition

As was shown in the previous examples, the role that each species plays in the ecosystem is an emergent property of our model of cooperative coevolution. Each species will focus on exploration until it finds something unique to contribute to the collective problem solving effort. Once it 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 for some other species to evolve individuals that perform the same function because they will receive no reward for doing so.

We have also suggested one possible algorithm for enabling an appropriate number of species in the ecosystem to emerge. This is accomplished as shown in figure 3.4 on the preceding page by creating new species when evolution stagnates and eliminating species that make no useful contribution. The emergent problem decomposition characteristics of cooperative coevolution will be addressed in much greater detail in chapters 5 and 6.

3.2.2 Interdependencies Between Subcomponents

In Lewis Carroll's (1871) classic children's tale *Through the Looking-Glass*, the Red Queen says to Alice:

> Now, *here*, you see, it takes all the running *you* can do, to keep in the same place. If you want to get somewhere else, you must run at least twice as fast as that!

The biologist Van Valen (1973) hypothesized that in natural ecosystems with interdependencies between species, the evolution of each species is being constantly driven by evolutionary changes in the species it interacts with. He called this the Red Queen's hypothesis because each species must constantly adapt just to remain in parity with the others.

The Red Queen's hypothesis also applies to species in our model of cooperative coevolution. Species representing interdependent subcomponents are coevolved within a single ecosystem where they can interact and coadapt. If a species does not make a "significant contribution" to the problem-solving effort it will eventually be eliminated. What constitutes a significant contribution from each species is constantly changing as a result of the adaptation of the other species in the ecosystem. Interdependencies between species and their constant adaptation to each other provide the engine of emergent problem decomposition.

Given that species are interdependent, an important related issue is how to model the patterns of interaction that occur between individuals of one species and those of another. In our model we usually assume a greedy strategy in which the current best individual from each species is chosen as the point of contact with all other species. That is, all the individuals from one species will be evaluated within the context of the best individual from each of the other species. We selected this strategy because it is simple and requires a minimal number of collaborations between individuals to be evaluated. We will see in chapter 4 that the strategy has some undesirable characteristics as well. In contrast, the patterns of interaction between interdependent species in nature can be extremely complex. Our greedy interaction strategy is not intended to be an accurate model of this process. Rather, it simplifies the environment and enables us to focus here on other aspects of coevolution. However, the design of more ecologically faithful computational models of the interaction between species is clearly a crucial topic for future research.

3.2.3 Credit Assignment

Credit assignment occurs in our model of cooperative coevolution at two distinct levels: the individual level and the species level. Credit assignment at the individual level must be performed during each reproductive cycle as shown in figure 3.2 on page 32, and is used to determine the likelihood that individuals will reproduce. Credit assignment at the species level is only performed when evolution stagnates, and is used to determine whether any species should be eliminated entirely from the ecosystem as shown in figure 3.4 on page 34.

When evaluating the fitness of individuals from one species, the representatives from the other species remain fixed as shown in figure 3.3 on page 33. 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 at some instant in time. Furthermore, by only assigning the fitness values to the individuals of the species being evaluated and not to the representatives of the other species who are providing context, the credit assignment problem is greatly simplified because there is no need to determine which species contributed what.

On the other hand, to determining whether species should be removed from the ecosystem when evolution stagnates, we do occasionally need to make a rough estimate of the level of contribution each makes. Although the precise way in which species contributions are computed is problem-specific, in some domains we can simply construct a collaboration consisting of representatives from all the species and temporarily withdraw them one at a time while measuring the change in fitness that occurs.

3.2.4 Population Diversity

Evolving genetically isolated species in separate populations eliminates the requirement for indefinitely maintaining sufficient population diversity to support coadapted subcomponents. The diversity within a species only needs to be maintained long enough for it to discover a useful niche not being exploited by some other species and to survive the period of rapid adaptation that takes place as a stable problem decomposition emerges. Generally, standard genetic operators and reasonable population sizes provide enough diversity for this to occur. When they are not sufficient, the creation of new species introduces additional genetic material into the ecosystem.

In a sense, the problem of maintaining population diversity has been transformed into a problem of maintaining diversity among the species in the ecosystem. While there is considerable evolutionary pressure for a single population to converge, no such pressure exists among genetically isolated species being evolved in separate populations. Instead, rewarding the individuals from each species based on how well they collaborate with representatives from the other species encourages them to make unique contributions to the problem-solving effort. This ensures that the ecosystem will consist of a diverse collection of species.

3.2.5 Parallelism

Our model of cooperative coevolution can take advantage of all the previous methods for parallelizing evolution algorithms. This includes running slave processes on separate processors to perform fitness evaluations, using the coarse-grain island model and distributing the population of a species across a few processors while allowing occasional migration, or using the fine-grain approach of distributing individuals belonging to a species across many processors while allowing them to interact with one another using localized mating rules. In addition, our model enables an additional coarse-grain method of parallelism, specifically, assigning a separate processor to each species.

The advantage, with respect to parallelism, of evolving genetically isolated species in separate populations is that each species can be evolved by its own semiautonomous evolutionary algorithm. 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. The reason this is advantageous can be explained by *Amdahl's*

Law (Amdahl 1967), which places an upper limit on the amount of speedup achievable as a function of the percentage of parallelizable code. Given the following definition of speedup:

$$S = \frac{\mathrm{T}(1)}{\mathrm{T}(N)},$$

where $\mathrm{T}(N)$ is the time required to solve the problem given N processors, Amdahl's Law can be expressed as

$$\max(S) = \frac{N}{\beta N + (1 - \beta)},$$

where N is the number of processors and β is the ratio of time spent executing parallel code to total execution time (Lewis and Hesham 1992). By reducing the reliance on centralized control structures and allowing the species to be evolved semiautonomously, the algorithm spends more time evolving species in parallel and less time serially managing that evolution—thus enabling speedup to approach more closely its absolute limit of 1.0. Of course, Amdahl's Law only applies when the semantics of the algorithm being parallelized are held constant with respect to N.

3.3 Additional Advantages of the Model

3.3.1 Speciation Through Genetic Isolation

We have already discussed two important advantages of speciation through multiple genetically isolated populations: the ease of maintaining diversity in the ecosystem and the high degree of parallelization of the model that can be achieved. Two additional advantages of speciation through multiple genetically isolated populations are the capability to evolve species with heterogeneous representations simultaneously, and the elimination of unproductive cross-species mating.

As we apply evolutionary computation to larger problems, the capability of coevolving subcomponents with heterogeneous representations will become increasingly important. 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. Multiple genetically isolated populations enable each of these components to be represented appropriately. As long as the components are evolved simultaneously in an environment in which they can interact, coadaptation is possible.

Regarding cross-species mating, imagine that our evolving autonomous robot is controlled by a wide variety of specialized species. For example, individuals of one species may implement a high-level planning component that prioritizes a list of tasks assigned to the robot, while individuals of another species determine how much pressure to apply to a set of manipulator jaws to enable the robot to pick up an object without crushing it. It is not likely that mating individuals from these two species with one another will produce anything viable. When individuals become highly specialized, mixing their genetic material through cross-species mating will usually produce non-viable offspring, as is demonstrated in nature. In natural ecosystems, 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 1983). Although the specific conditions under which speciation occurs are a matter of debate, clearly species differences are sustained over time in large part due to this lack of interbreeding.

3.3.2 Generality

Nature shows us that evolution is the paragon of general-purpose problem solving, and it is important to preserve this generality in our computational models of the process. The model of cooperative coevolution is applicable to a wide range of decomposable problems. Our focus in this dissertation is on solving problems as diverse as those from the domains of function optimization, concept learning, and artificial neural network construction. In addition, the model is not limited to any particular representation or underlying evolutionary algorithm. We will later demonstrate that the model can extend the usefulness of both genetic algorithms and evolution strategies. It is even possible to mix evolutionary algorithms in the same system, as in evolving one species having a binary string representation with a genetic algorithm and coevolving another species having a real-valued vector representation with an evolution strategy. The use of such heterogeneous coevolutionary algorithms is an area for future research.

3.3.3 Efficiency

Along with the before-mentioned efficiency one can achieve by taking advantage of the high degree of parallelism possible with the model, there is another important source of efficiency. By evaluating individuals from one species within the context of a subset of individuals from other species, the search becomes highly constrained.

A problem whose solution is decomposed into n interdependent subcomponents, each of which is represented by a binary string of length k, will have a solution space of size $(2^k)^n$. If we evaluate individuals from one species within the context of a single representative from each of the other species, we constrain the search to a series of regions of size 2^k. For example, a problem consisting of five subcomponents, each represented by a 32-bit binary string, would have a solution space size on the order of 10^{48}. If each of these subcomponents were represented as a species and evolved for 100 ecosystem generations[1] under the constraint just described, an upper bound on the order of 10^{12} would be placed on the size of the solution space searched. This form of search space constraint is similar to the coordinate strategy that has been commonly used in the field of traditional function optimization to solve high-dimensional problems. Of course if the subcomponents are interdependent, we are not guaranteed to find the global minimum or maximum when utilizing this form of constraint (Schwefel 1995). The issue of constraining the search space will be explored in greater detail in chapter 4.

3.4 A Simple Example

We conclude this chapter by instantiating the cooperative coevolutionary model and applying it to the simple string covering task described in example 2.4 beginning on page 16. A complete listing of the Lisp program code for this instantiation is included in appendix A.

[1]Recall that an ecosystem generation is the time required for all the species to complete a single generation.

Recall that in the string covering task we are given a set of six binary strings called the target set, and the goal is to find the best possible three-element set of matching strings called the match set. The match strength between two strings is determined by summing the number of bits in the same position with the same value. Here we use basically the same target set as in the original example; however, to make the problem a little more challenging we repeat the pattern of each of the four-bit strings to increase their lengths to 32 bits. Generalization is required to solve this problem since it is obviously impossible to cover six distinct target strings completely with three match strings.

For this example, we choose to use a genetic algorithm to evolve each of the species and we let each individual directly represent one of the 32-bit match strings; that is, we make no distinction between the genotype and the phenotype of an individual. The populations of each of the species are initialized randomly. We arbitrarily set the population size to 50, the genetic operators to two-point crossover at the rate of 0.6 and bit-flipping mutation at a rate equal to the reciprocal of the chromosome length, and the selection strategy to a scaled fitness proportionate scheme.

As in the original example, we decompose the problem into three subtasks, each consisting of finding the best value for one of the three match strings. This is mapped into our coevolutionary model by assigning a different species to each of the three subtasks. We add a new species to the ecosystem each generation until all three subtasks are accounted for. Since the problem specifies a match set of size three, there is no reason to further model the creation of new species and the extinction of those that are non-viable. This is not a complete hand-decomposition of the problem because we provide no information to the system concerning which target strings each species should cover. We constrain the search space by selecting only the current best individual from each species as its representative.

To evaluate the fitness of an individual from one of the species, we first form a three-element match set consisting of the individual in question and the representatives from the other two species. Next, the match strength between each string in the target set and each string in the match set is computed. The following linear function, which simply sums the bits in the same position with the same value, is used to determine the match strength between two strings, x and y, of length l:

$$\text{strength}(\vec{x}, \vec{y}) = \sum_{k=1}^{l} \begin{cases} 1 & \text{if } x_k = y_k \\ 0 & \text{otherwise.} \end{cases} \tag{3.2}$$

Although the match strengths of all possible pairs of strings in the two sets are computed, only the largest match strength for each target string is retained. The final step in the fitness computation is to average the retained strengths. Formally, the fitness equation is as follows:

$$\text{fitness} = \frac{1}{n} \sum_{i=1}^{n} \max(\text{strength}(\vec{x}_i, \vec{y}_1), \ldots, \text{strength}(\vec{x}_i, \vec{y}_m)), \tag{3.3}$$

where x is a target set element, y is a match set element, n is the target set length, and m is the match set length. Two points need to be emphasized here. First, information concerning which match string produced the strongest match for a particular target string is not used in the fitness computation. We are only concerned with the strength of the collaboration as a whole—not with who contributed what. Second, it may be that the

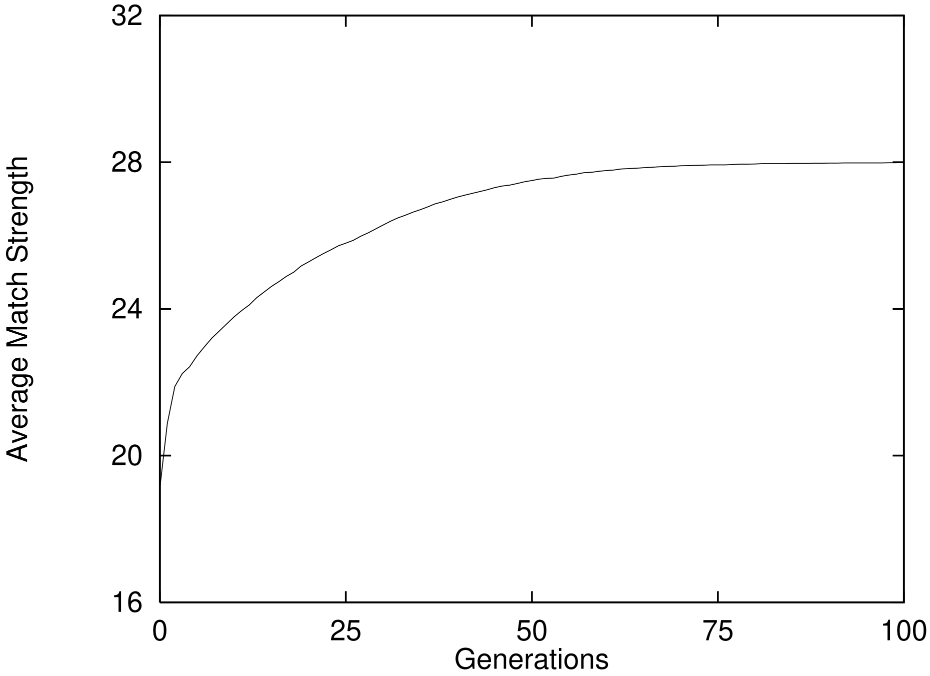


Figure 3.5: Average match score between target set and best collaborations

match string represented by the individual being evaluated does not produce the strongest match for any of the strings in the target set; that is, it makes no contribution to the problem-solving effort whatsoever. If this is the case, its fitness will simply reflect the fitness of the representatives from the other two species. After the fitness values of all members of a species are computed, the differences between the values reflect individual contributions because everyone is evaluated within the same context.

The graph plotted in figure 3.5 shows the fitness of the collaboration formed by the current best individual from each species at the end of each generation averaged over 100 runs. The expected fitness of a randomly initialized match string given a set of 32-bit target strings is 16.0. The initial average match strength of 19.1 is slightly greater than the expected value because it reflects the fitness of the best member of a population of 50 match strings. Recall that the ecosystem initially consists of a single species; therefore, the initial match set contains only one element. As the figure shows, the fitness of the collaborations quickly increases as the remaining two species are added to the ecosystem over the course of the next two generations. Improvement then slightly slows but continues to approach asymptotically a fitness of 28.0 until the final generation. Although the best possible fitness is 32.0—produced when each 32-bit target string is matched perfectly—it is clearly not possible to achieve this level when we only have three match strings to cover six distinct target strings.

The graph plotted in figure 3.6 on the following page is generated only from the initial run and shows the amount each species is contributing to the problem-solving effort. The individual contributions are computed to provide us with additional insight into the macroevolutionary dynamics of the system. We emphasize that this information is not used

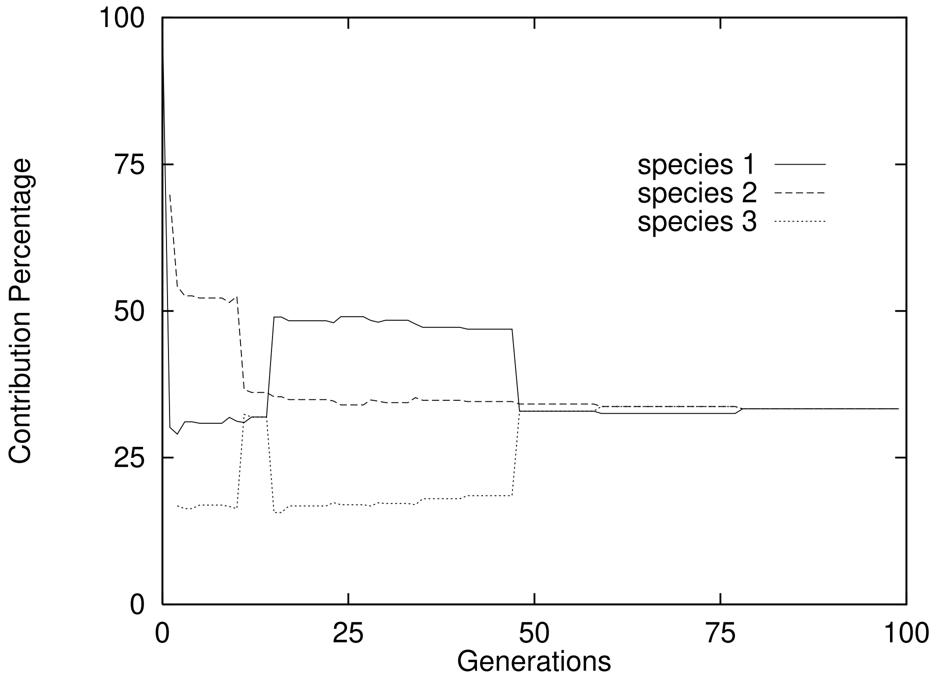


Figure 3.6: Percent contribution of each species to best collaborations

in any way by the evolutionary process. As in the previous graph, we produced this plot by forming a collaboration consisting of the current best individual from each species at the end of each generation and computing the match strength between the elements of the target set and the members of the collaboration. To measure the contribution of one of the species, we summed the subset of strengths for which it produced a better match than either of the other two representatives. More formally and ignoring ties, the contribution function is defined as follows:

$$\text{contribution}(\vec{r}\,) = \frac{1}{n} \sum_{i=1}^{n} \begin{cases} \text{strength}(\vec{r}, \vec{y}_i\,) & \text{if best match} \\ 0 & \text{otherwise,} \end{cases} \tag{3.4}$$

where r is the individual chosen to represent its species in the collaboration and $\vec{y}_i$ is a string from the n-element target set. In the actual contribution computation, ties were broken randomly, although in practice they rarely occurred. The graph plots the contribution of each species as a percentage of the total strength of the collaboration; therefore, the three curves in figure 3.6 always sum to 100 percent. The figure clearly shows a period of relative instability in the contribution made by each of the species during the early generations. This is when the system is constructing a reasonably good problem decomposition, or using evolutionary terminology, the period in which the species are acquiring stable niches. In the particular run plotted, the contributions do not significantly change after generation 48. The emergent decomposition properties of the model of cooperative coevolution will be studied in greater depth in chapters 5 and 6.

Chapter 4

ANALYSIS OF SENSITIVITY TO SELECTED PROBLEM CHARACTERISTICS

In this chapter, we explore the robustness of our computational model of cooperative coevolution with respect to selected problem characteristics that potentially will have a negative effect on the performance of the model. We also suggest possible approaches to overcoming any exposed difficulties. We defer the issue of determining an appropriate decomposition until the next chapter and here assume a static hand-decomposition of the problem.

4.1 Selection of Problem Characteristics

Three characteristics of decomposable problems that we have identified as likely to have a significant effect on the performance of cooperative coevolution are:

1. the amount and structure of interdependency between problem subcomponents,

2. the number of subcomponents resulting from a natural decomposition of the problem, and

3. the degree of accuracy possible when evaluating the fitness of collaborations among the subcomponents.

While these are certainly not the only characteristics capable of affecting the model, by referring to the detailed description of cooperative coevolution in the previous chapter we will now justify the choice of each for inclusion in an initial sensitivity analysis.

To understand the relevance of the first characteristic, recall from figure 3.2 on page 32 that each individual is evaluated by first forming a collaboration with representatives from each of the other species in the ecosystem. If the species represent independent problem subcomponents, the choice of partners in these collaborations is irrelevant—each species might as well be evaluated in isolation. However, if the species are interdependent, evolving one will warp the fitness landscapes associated with each of the other species to which it is linked. Therefore, the amount and structure of this interdependency are likely to play a major role in the performance of the model. The second characteristic is also an obvious choice as the scalability of an approach is often an important consideration. From figure 3.3 on page 33 it is clear that as the number of subcomponents (species) increases, the patterns of interaction within the domain model will likely become more complex.

This increase in complexity may in turn have a significant impact on the performance of cooperative coevolution. The third characteristic is perhaps a less obvious choice. Recall from figure 3.1 on page 31 that a new representative is chosen from each species at the end of every generation. The particular strategy used for choosing these representatives in most of our experiments is simply to select the individual from each species with the highest fitness. Any inaccuracy in evaluating the fitness of population members is compounded; first, in the misallocation of reproductive cycles to weak members of the respective populations; and second, in a poor choice of representatives, which will distort the evaluation of all the collaborations the representatives participate in. Whether this will negatively affect on the performance of the coevolutionary model is a question that we will answer later in this chapter.

4.2 Methodology

It has become common practice in the field of evolutionary computation to compare algorithms using large test suites. This methodology has been especially prevalent among those whose focus is the application of evolutionary computation to function optimization; see, for example, (Gordon and Whitley 1993). In turn, this has lead to arguments concerning the merits of commonly used test functions and suggestions for building better test suites (Whitley, Mathias, Rana, and Dzubera 1995; Salomon 1996). Although it is certainly important to understand the principles of designing good test functions, one should not consider these functions simply as weights to be placed on a balance. If the balance tips to the right, algorithm A is better than algorithm B; if it tips to the left, algorithm B is better. The "no free lunch" theorem (Wolpert and Macready 1995) proves that when comparing two search methods, no matter how many objective functions tip the balance in favor of one approach, an equal number exist for tipping the balance the other way. A consequence of this is that given all possible objective functions, the average performance of any two search algorithms is identical. More formally, given a pair of algorithms a_1 and a_2,

$$\sum_f \Pr(\vec{c} \mid f, m, a_1) = \sum_f \Pr(\vec{c} \mid f, m, a_2), \tag{4.1}$$

where $\vec{c}$ is a histogram of fitness values resulting from the evaluation of a population of m distinct points generated by an algorithm, a, on some objective function, f. In words, the equation states that the conditional probability of obtaining a particular histogram when summed over all possible objective functions is exactly the same, regardless of the algorithm used to generate the population. Therefore, attempting to design the perfect all-inclusive test suite for determining whether one algorithm is "better" in a general sense than another is futile. One algorithm only outperforms another when its biases with respect to the search space better match the specific objective function being optimized. More properly, when evaluating a new algorithm one should seek an understanding of the specific problem characteristics that the algorithm will use to its advantage and those that will obscure, inveigle, or obfuscate.

The methodology we adopt here is to perform a sensitivity analysis on the three characteristics of decomposable problems identified above as being likely to have a major effect on the performance of the coevolutionary model. For each characteristic, comparisons will be made between a coevolutionary and a standard evolutionary algorithm that differ only in whether they utilize multiple species. All other aspects of the algorithms are equal and are held constant over each set of experiments. The standard evolutionary algorithm provides a reference point from which to measure the amount of effect directly attributable to coevolution. Through focused experimentation using tunable test functions chosen specifically to emphasize these three characteristics, we hope to gain insight into the circumstances under which they will have a negative impact on the performance of the model and how any exposed difficulties may be overcome.

4.3 Sensitivity to Random Epistatic Interactions

The first characteristic we will investigate is the interdependency between problem subcomponents. This characteristic is more complex than the other two. Not only do we need to be concerned with the amount of interdependency, but its structure is also important. By *structure*, we mean the relationship between fitness dependencies within the genotype space. In the field of genetics these fitness dependencies are called *epistatic interactions*. Technically, *epistasis* refers to the fitness linkage between multiple genes. Linkages can occur between genes in the same species and between genes in two or more species.

Before we explain what we mean by *random* epistatic interactions, we first present an example of the complex combination of linkages that often occurs among species in nature—Batesian mimicry in African swallowtail butterflies (Turner 1977; Smith 1989). In Batesian mimicry, one species of butterfly that is palatable will resemble another butterfly species that is unpalatable. The palatable species *Papilio memnon*, for example, mimics other unpalatable African species through the shape, color, and pattern of its wings, and its abdomen color. These characteristics are determined by a number of different genes within the species. Linked genes such as these within a single species are collectively referred to as a *supergene*. Of course, the fitness of *P. memnon* depends on genes controlling both the appearance and palatability of the butterfly species it mimics. As the frequency of *P. memnon* in the population increases, the effectiveness of the mimicry decreases as predator species begin to associate its appearance with a palatable rather than an unpalatable butterfly. This reduces the fitness of both *P. memnon* and the truly unpalatable species it mimics.

The Batesian mimicry example makes an important point. The structure of epistatic interactions in ecosystems from nature tends to be quite complex and difficult to understand. In the case just described, there are many genes involved, linkages occur both within and between the two species, and the actual result of the mimicry is sensitive to a variety of factors such as the population densities of the butterflies and the status of butterfly predators sharing the ecosystem. As a result of this complexity, a random energy model has been used to capture the statistical structure of such systems (Kauffman 1989; Kauffman and Johnsen 1991; Kauffman 1993). That is, if two genes are linked, the effect of that linkage on the fitness of the organism will be random. We use the expression *random epistatic interactions* when referring to this type of linkage. In other domains, the interdependencies between problem subcomponents are more highly ordered. For example, in the field of real-valued function optimization the interaction between variables often forms a geometrical lattice of peaks and basins in the fitness landscape. These function variable interactions are analogous to the epistatic interactions that occur between genes. In this section we investigate random epistatic interactions and defer an investigation of highly ordered interactions until section 4.4. It is likely that problem subcomponents from domains inspired by nature, such as machine learning, will share the propensity of species from natural ecosystems to display complex interdependencies best captured by a random energy model.

There is another type of structure in epistatic interactions that we are less interested in. This is the relationship between linked genes and their position in the chromosome. Kauffman (1989) studied two different possibilities: a model in which linked genes have a random positional relationship, and one in which linked genes are always nearest neighbors. He showed through experimentation using a hill climbing algorithm[1] that it makes little difference which of these models is used with respect to the mean fitness of local optima and the length of adaptive walks from randomly chosen genotypes to local optima. Although we do not know how severe an effect the positional relationship between linked genes will have on the performance of evolutionary algorithms using position dependent operators such as two-point crossover, given that all the models being compared in this chapter use the same genetic operators, our assumption is that the relative performance differences attributable to this characteristic will be minimal. Therefore, we do not address this issue further and exclusively use the nearest neighbor gene linkage model in our experiments.

4.3.1 NK-Landscape Problem

The test problem we use in this experimental study of the effect of random epistatic interactions is a search for the global optimum within the context of Kauffman's (1989) tunable NK model of fitness landscapes. This is a random energy model, similar to a *spin glass* (Edwards and Anderson 1975), that is designed to capture the statistical structure of the rugged multipeaked fitness landscapes that we see in nature. Kauffman's motivation in creating the NK model was the study of coevolution and the conditions under which *Nash equilibria* will be attained. Nash equilibria are states in which all interacting species are optimal with respect to each other (Nash 1951).

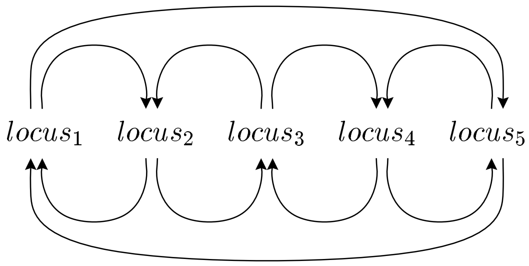
In the NK model, N represents the number of genes in a haploid chromosome and K represents the number of linkages each gene has to other genes in the same chromosome. An example of an NK model with $N = 5$ and $K = 2$ is shown in table 4.1. The linkages are displayed in the top portion of the table. For the purpose of the nearest neighbor gene linkage model, the chromosome forms a torus. Therefore, the gene at the first locus is linked to the genes at the last locus and the second locus; the gene at the second locus is linked to the genes at the first locus and the third locus; and so on. The bottom portion of the table shows the fitness contribution of each locus as determined by the allele of the corresponding gene and the alleles of the two genes to which it has linkages. The size of the contribution table grows exponentially as the number of gene linkages increases. Specifically, given a two-allele model, the table will have 2^{K+1} rows and N columns.

To compute the fitness of the entire chromosome, the fitness contribution from each

[1] Beginning with a randomly chosen genotype, the hill climbing algorithm successively moves to fitter single-locus variants.

Table 4.1: NK-landscape model for N=5 and K=2

Linkages



	Contributions				
Substring	$locus_1$	$locus_2$	$locus_3$	$locus_4$	$locus_5$
000	.968	.067	.478	.910	.352
001	.933	.654	.021	.512	.202
010	.940	.204	.379	.793	.288
011	.267	.357	.128	.703	.737
100	.803	.915	.511	.762	.456
101	.471	.300	.613	.073	.498
110	.220	.041	.565	.698	.951
111	.917	.630	.605	.938	.143

locus is averaged as follows:

$$f(chromosome) = \frac{1}{N}\sum_{i=1}^{N} f(locus_i),$$

where each locus fitness contribution, $f(locus_i)$, is selected from the appropriate row and column of the table. The table entries are generated by drawing randomly from a uniform distribution ranging from 0.0 to 1.0. In this example, each gene may occur as one of two alleles—either a zero or a one. Therefore, given $K = 2$, all possible values of a gene tuple formed from a target gene and the genes it is linked to are specified by the column of binary substrings shown in the table. Take, for example, the gene at the second locus. If this gene and both genes to which it is linked[2] were all of allele zero, the contribution of the second locus would be 0.067. If instead, the gene at the third locus was of allele one, the contribution of the second locus would be 0.654. In other words, the contribution of the second locus changes, even though the allele of the second gene remains the same.

Several observations can be made from Kauffman's studies of the NK model. As K increases, the number of peaks in the fitness landscape increases and the landscape becomes more rugged. By *rugged* we mean that there will be a low correlation between the fitness and similarity of genotypes, where the similarity metric used is Hamming distance. The extreme

[2]The gene at the second locus is linked to the genes at the first and third loci.

case of $K = 0$ produces a highly correlated landscape with a single peak, while the other extreme case of $K = N-1$ produces a landscape that is completely uncorrelated and has very many peaks. Another interesting observation is that as both N and K increase, the height of an increasing number of fitness peaks falls towards the mean fitness. This phenomenon, which Kauffman refers to as a "complexity catastrophe", is a result of conflicting constraints among the genes. From a function optimization perspective, searching for peaks with high fitness in this case is analogous to looking for a needle in a haystack.

The NK model shown in table 4.1 on the preceding page only supports gene linkages within a single chromosome. However, the complete NK model also supports the coupling of fitness landscapes from multiple species. To accomplish this, Kauffman adds a third parameter, C, that specifies the number of gene linkages between pairs of species (Kauffman and Johnsen 1991). In our implementation, the first C genes from each species are the ones chosen to affect other species. Therefore, if we add the parameter $C = 3$ to the example above and introduce a second species, each gene of species A would be linked to its two neighboring genes and the first three genes of species B. Similarly, each gene of species B would be linked to its two neighboring genes and the first three genes of species A.

4.3.2 Experimental Results

Since we are using a two-allele model, the chromosomes of individuals whose phenotypes are points on an NK landscape can be represented with binary strings of length N. As in the string covering example described in section 3.4, we use a coevolutionary genetic algorithm to evolve these individuals. In all experiments, we initialize the populations of each species randomly, use a population size of 50, a two-point crossover rate of 0.6, a bit-flipping mutation rate set to the reciprocal of the chromosome length, fitness proportionate selection, and balanced linear scaling. Unless otherwise noted, fitness curves are generated from an average of 100 runs.

We begin by investigating the effect of random epistatic interactions within a single 24-bit chromosome on a standard genetic algorithm. This is simply our coevolutionary implementation restricted to a single species. The graph in figure 4.1 on the next page shows the fitness of the best individual seen so far over a period of 500 generations for various levels of epistasis ranging from none to moderate. A comparison between the graph and the expected global optimum values[3] shown in table 4.2 reveals that the genetic algorithm easily finds the global optimum when epistasis is low. However, when epistasis is increased to a moderate level by setting K to 7, adaptation slows dramatically.

In figure 4.2 on page 50 we compare the standard genetic algorithm with random search[4] at the extremes of no epistasis ($K = 0$) and maximum epistasis ($K = 23$). While the genetic algorithm is far superior to random search when epistasis is low, at the maximum level random search actually outperforms genetic search. This should not be too surprising, considering that the fitness landscape at the maximum level of epistasis is completely uncorrelated. Therefore, the primary bias of genetic search—the allocation of an exponentially

[3]These values were computed experimentally by enumerating the entire space of 100 different randomly generated landscapes for each value of K. The 95-percent confidence ranges of all the expected optimum values are within ± 0.001 of the mean.

[4]Random search draws genotypes from a uniform distribution with replacement. In the figure, one generation of random search performs the same number of evaluations as a generation of genetic search.

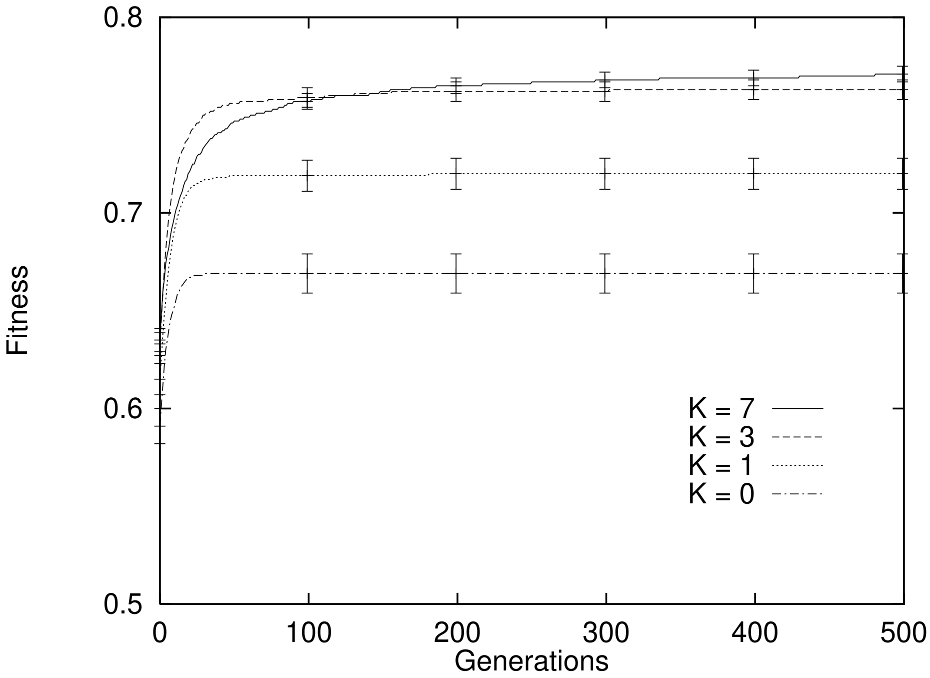


Figure 4.1: Standard genetic algorithm applied to 24-bit NK landscape with various levels of epistasis

Table 4.2: Expected global optimum of 24-bit NK landscapes

K	*Expected optimum*
0	0.667
1	0.712
3	0.751
7	0.778
15	0.794
23	0.800

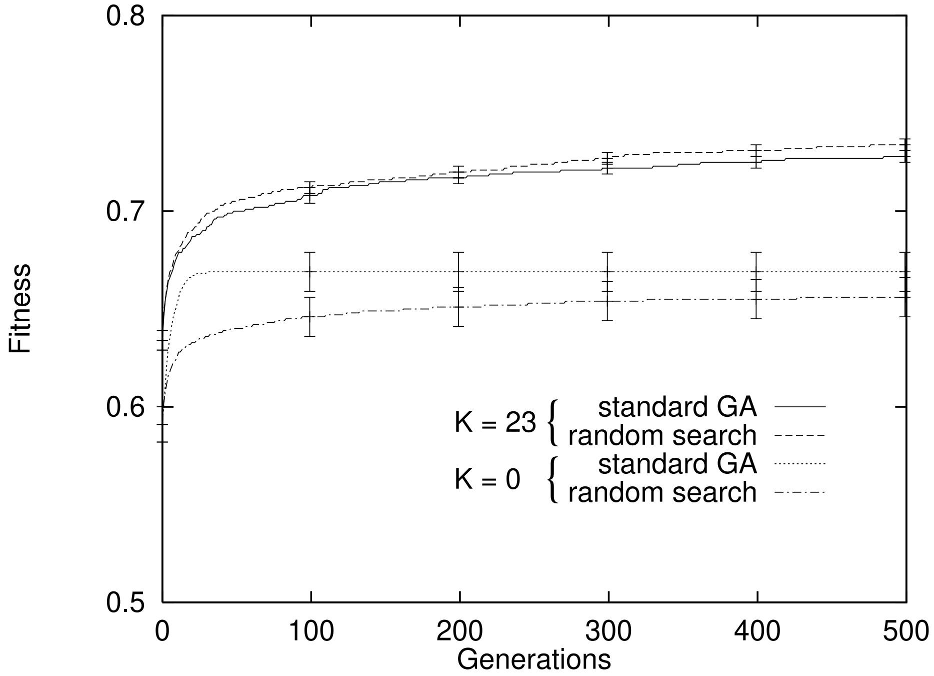


Figure 4.2: Standard genetic algorithm and random search on 24-bit NK landscape with no epistasis (K = 0) and maximum epistasis (K = 23)

increasing number of trials to observed regions of the solution space with above average fitness—produces absolutely no benefit and wastes computational energy by increasing the likelihood of reevaluating the same points. Another observation is that neither search algorithm has come close to finding the expected global optimum of 0.800 at the maximum level of epistasis after 500 generations (25,000 evaluations). We pointed out earlier that as both N and K increase, the height of an increasing number of fitness peaks falls towards the mean fitness, which in this case is 0.5. A consequence of this is that the landscape is densely populated by average individuals. This and the fact that the solution space is of size 2^{24} explain why it is not likely that an individual with a fitness close to the global optimum will be found in only 500 generations.

In the next set of experiments, we compare the effect of various levels of intraspecies epistasis on the relative performance of a coevolutionary genetic algorithm and a standard genetic algorithm. In this study, we simultaneously search for the optimum on two separate 24-bit NK landscapes, which we will refer to as landscape A and landscape B. The standard genetic algorithm represents solutions to this problem as a single 48-bit chromosome. Specifically, the first half of the genotype represents a point on landscape A, and the second half of the genotype represents a point on landscape B. In contrast, the coevolutionary genetic algorithm evolves a species of 24-bit individuals for landscape A and a separate species of 24-bit individuals for landscape B. There are no interactions—epistatic or otherwise—between individuals from the two species. Therefore, in a literal sense no coevolution is occurring. The "coevolutionary" genetic algorithm in this case is equivalent to running two non-communicating standard genetic algorithms in parallel.

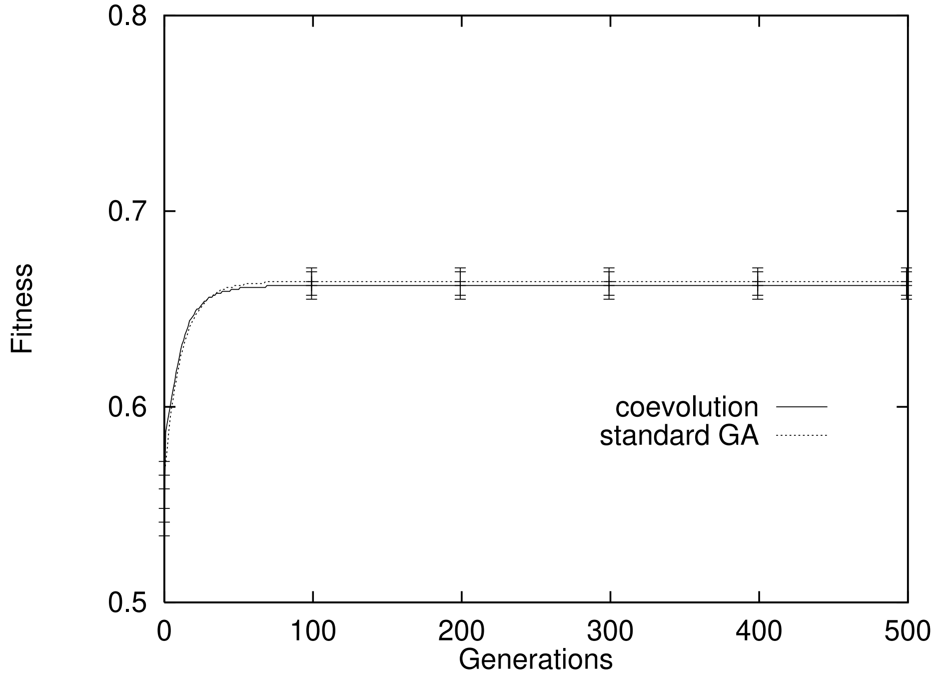


Figure 4.3: Coevolution and standard genetic algorithm on two uncoupled 24-bit NK landscapes with no epistasis (K = 0)

When there is no epistasis within or between species, as shown in figure 4.3, both algorithms easily find the global optimum of both landscapes and are statistically equivalent in performance. This also holds for a low level of epistasis within the species ($K = 3$) as shown in figure 4.4. However, when intraspecies epistasis is increased to a moderate level by setting K to 7 as shown in figure 4.5 on the next page, the performance advantage of coevolution over the standard genetic algorithm becomes obvious. Continuing this trend, when intraspecies epistasis is maximized by setting K to 23 as shown in figure 4.6, the performance advantage of coevolution over the single chromosome model becomes even more significant. A likely explanation for the superior performance of coevolution is that genetic isolation and independence ensures the evaluation of individuals from one species is not corrupted by the evaluation of individuals from the other species. When a single chromosome represents points on both landscapes, as is the case with the standard genetic algorithm, the high fitness of one point can easily be masked by its association with a point on the other landscape having low fitness. This masking effect becomes more likely as the probability of genetic operators changing a highly fit genotype into a weak genotype increases with higher values of K.

In the final set of experiments in this section we investigate the effect of random epistatic interactions between species. As in the previous study, we simultaneously search for the optimum on two separate 24-bit NK landscapes. However, here we fix the intraspecies epistasis at a moderate level by setting K to 7, and vary the C parameter that controls the number of linkages each gene of one species will have to genes of the other species.

Once we add the additional complexity of linkages between species, the coevolutionary genetic algorithm is no longer equivalent to running two non-communicating standard ge-

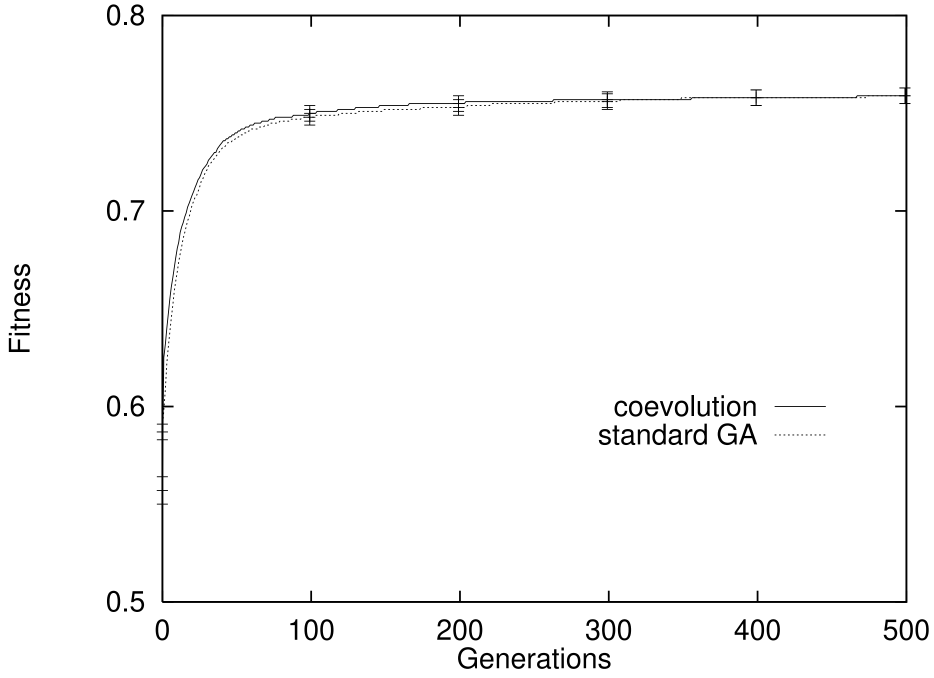


Figure 4.4: Coevolution and standard genetic algorithm on two uncoupled 24-bit NK landscapes with low epistasis (K = 3)

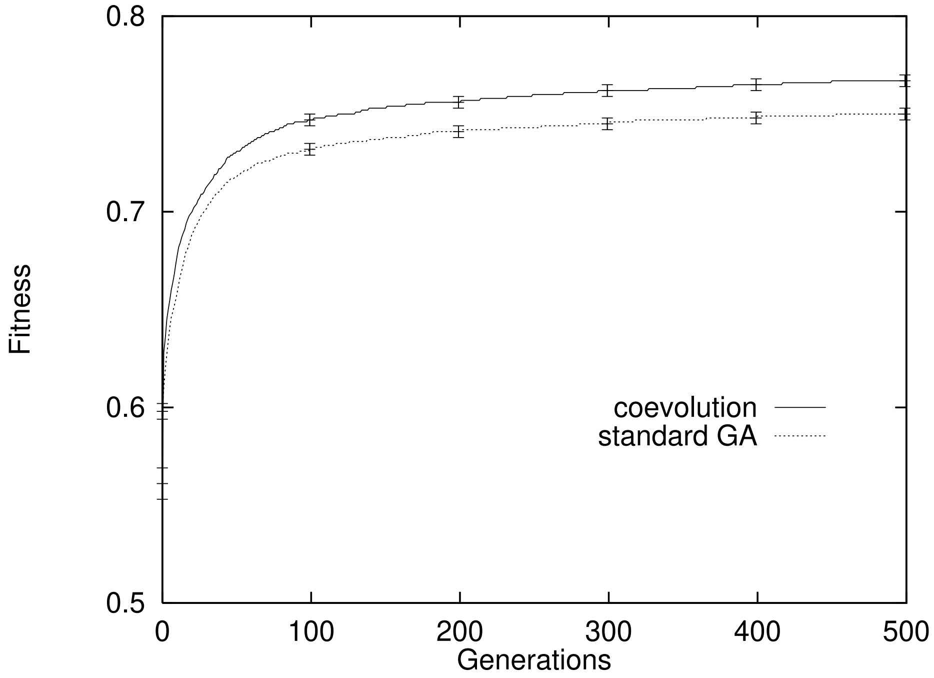


Figure 4.5: Coevolution and standard genetic algorithm on two uncoupled 24-bit NK landscapes with moderate epistasis (K = 7)

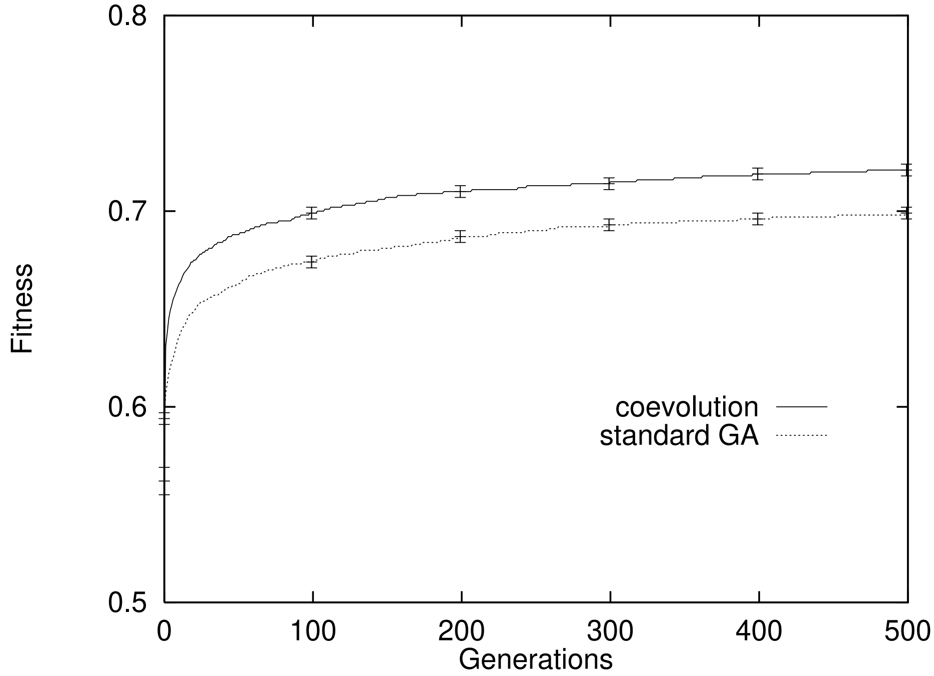


Figure 4.6: Coevolution and standard genetic algorithm on two uncoupled 24-bit NK landscapes with maximum epistasis (K = 23)

netic algorithms in parallel. As the number of linkages between species increases, there is a corresponding increase in the likelihood that one species will warp the fitness landscape associated with the other as it is evolved. Therefore, if we evolve the species in isolation and bring them together at some point in the future, we will usually see an abrupt drop in fitness when they are merged. The probability of a fitness drop is a function of the amount of linkage between species. This is illustrated graphically in figure 4.7 on the following page for various levels of interspecies epistasis. With C set to only 4, the landscapes are warped so severely that the fitness after merging the solutions drops nearly to 0.5, which is the mean fitness of each landscape. These results were generated from the average of 200 runs of two species evolved independently and merged after the completion of 500 generations.

In figures 4.8 through 4.11 beginning on the next page we compare the effect of increasing levels of interspecies epistasis on the relative performance of a coevolutionary genetic algorithm and a standard genetic algorithm by varying C from 2 to 16. As before, the standard genetic algorithm represents solutions to this problem as a single 48-bit chromosome and the coevolutionary genetic algorithm evolves a separate species of 24-bit individuals for each landscape. From these figures we see that while increasing random epistatic interactions between species clearly increases the problem difficulty, it has little effect on the relative performance of the two models. Although we will see in the next section that this observation does not necessarily hold when the epistatic interactions are highly ordered, it is nonetheless an extremely important result given our hypothesis that problems from domains inspired by nature are likely to have complex interdependencies between their subcomponents that are characteristic of the random epistatic interactions between NK landscapes.

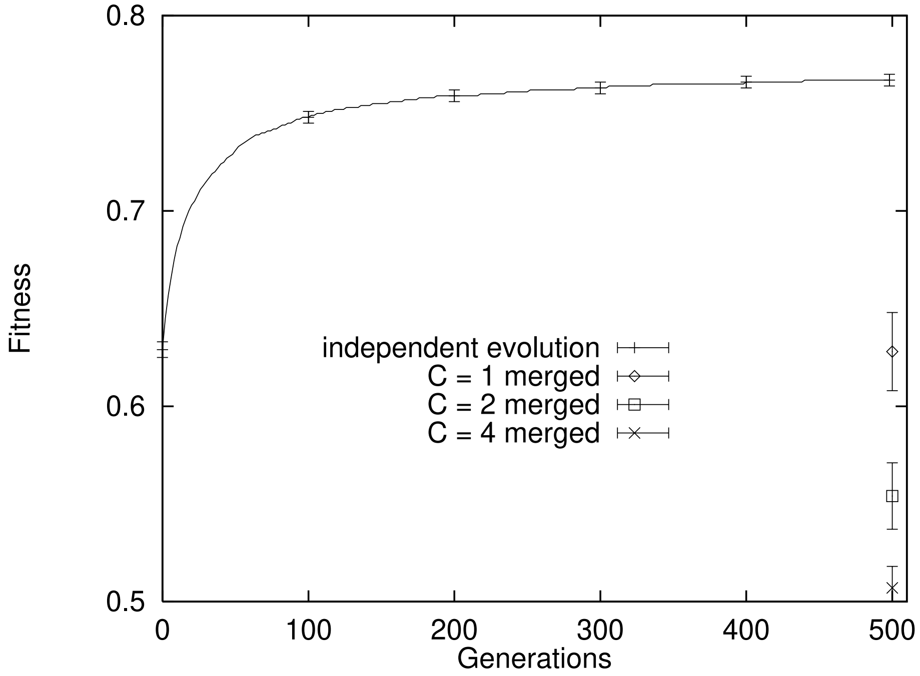


Figure 4.7: Effect of optimizing coupled NK landscapes separately and merging the final solutions

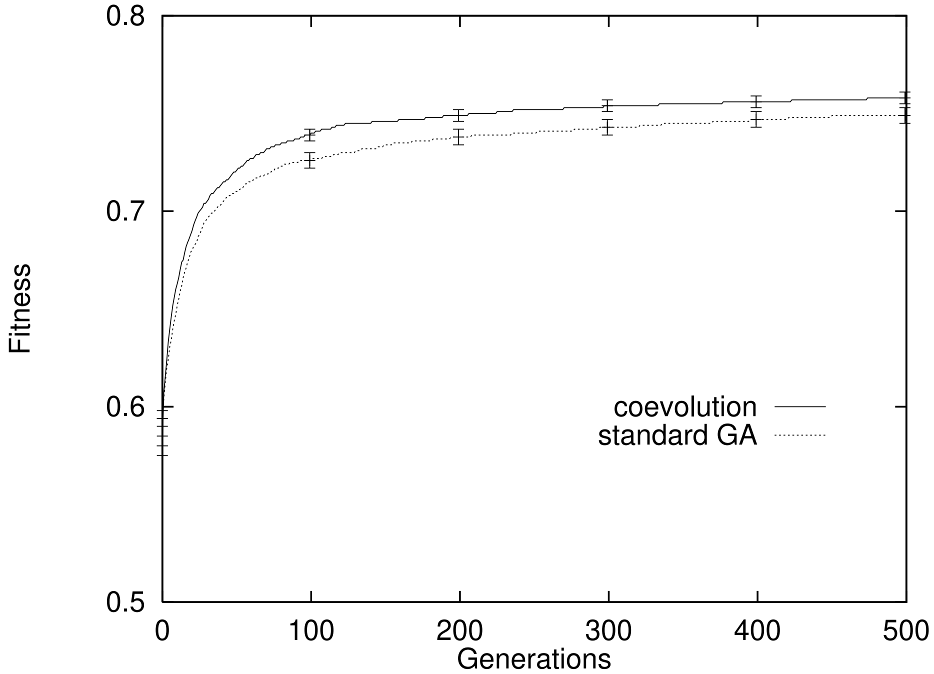


Figure 4.8: Coevolution and standard genetic algorithm on two coupled 24-bit NK landscapes (K = 7 and C = 2)

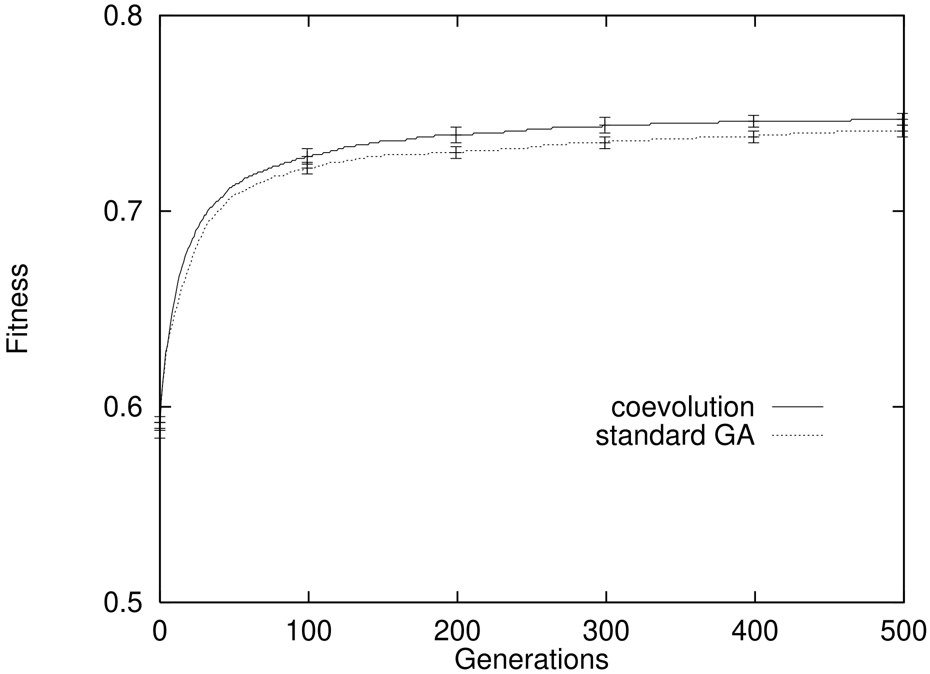


Figure 4.9: Coevolution and standard genetic algorithm on two coupled 24-bit NK landscapes (K = 7 and C = 4)

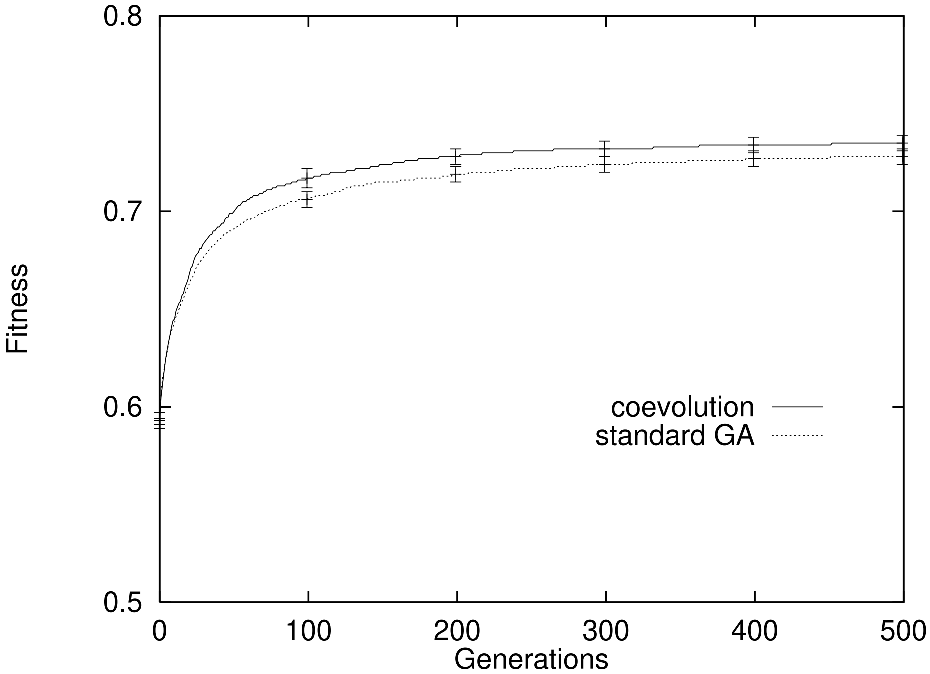


Figure 4.10: Coevolution and standard genetic algorithm on two coupled 24-bit NK landscapes (K = 7 and C = 8)

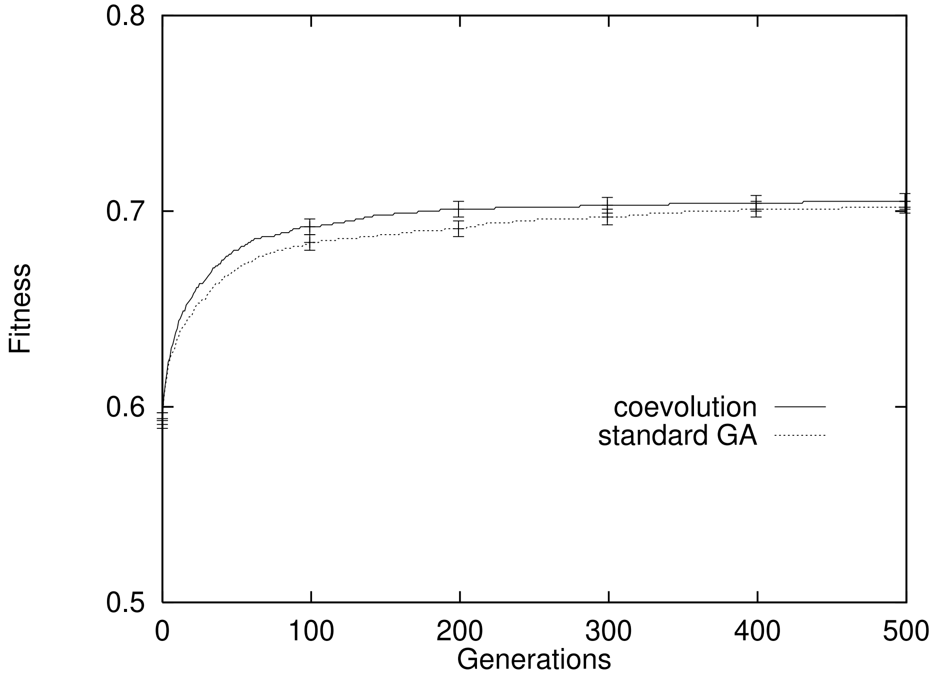


Figure 4.11: Coevolution and standard genetic algorithm on two coupled 24-bit NK landscapes (K = 7 and C = 16)

4.4 Sensitivity to Highly Ordered Epistatic Interactions

Now that we have an understanding of the effect of random epistatic interactions on the coevolutionary model, we investigate highly ordered epistatic interactions. A domain where we find highly ordered interactions between problem subcomponents is real-valued function optimization. Specifically, we are referring to the interaction between function variables, which often forms a geometrical lattice of peaks and basins in the fitness landscape. Because function variables are analogous to genes, we will continue to use the biological term *epistatic interactions* when referencing their interdependencies.

4.4.1 Coevolutionary Function Optimization

The application of evolutionary computation to function optimization has a rich history, beginning with the initial evolution strategy research by Rechenberg (1964) whose work was motivated by the need to solve difficult parameter optimization problems in the field of aerodynamics. Much genetic algorithm research has also been motivated by the need to solve difficult optimization problems. Although the original genetic algorithm work by Holland (1975) was motivated instead by a desire to build *adaptive* systems, these algorithms for a time became so closely coupled to the domain of function optimization that many began thinking of them only in terms of how well they performed this task. Interestingly, a paper titled "Genetic Algorithms are NOT Function Optimizers" was published by De Jong (1993), whose 1975 dissertation was the catalyst of the genetic algorithm based function optimization movement.

Given that a solution to a function optimization problem consists of a vector of n variable values, a natural decomposition is to coevolve n species, each of which is composed of a population of individuals representing competing values for a particular variable (Potter and De Jong 1994). Cooperative coevolution begins by initializing a separate species for each function variable. In our experiments, each individual consists of a 16-bit binary string. The fitness of an individual is computed by combining it with a representative from each of the other species, converting the binary strings in the resulting collaboration to a vector of real values, and applying the vector to the target function. Initially, no fitness information is available to intelligently choose the species representatives, so random individuals are selected. However, once a population associated with a species has been completely evaluated, the current best individual rather than a random individual is elected to represent the species in future collaborations.

This method of evaluating the fitness of alternatives for a variable value represented by individuals in one species by combining each of them with the current best variable values from the other species is in effect searching along a line passing through the best point on the fitness landscape found so far. As previously mentioned, this method of constraining the search space is similar to the relaxation method used in traditional parameter optimization (Southwell 1946; Friedman and Savage 1947). Although the method has some potential problems as we will see in section 4.4.4, it gives us a starting point for future refinements.

The standard genetic algorithm we use for comparison simply represents the vector of n variable values with a binary chromosome of length $n \times 16$, and evolves alternative vectors in a single population.

4.4.2 Function Separability

Closely related to the structure of epistatic interactions in function optimization is the notion of separability. If an objective function of n variables is *separable*, it can be rewritten as a sum of n single-variable objective functions (Hadley 1964). The general form of a separable function is expressed by the following equation:

$$f(x_1, \ldots, x_n) = g_1(x_1) + g_2(x_2) + \cdots + g_n(x_n).$$

Using the coevolutionary model to solve a separable objective function is roughly equivalent to running multiple non-communicating standard genetic algorithms in parallel, each of which is responsible for evolving the optimum value for a single variable. This is quite similar to the previous NK-landscape experiments in which there were no linkages between species. In both cases, the partial objective functions can be solved independently.

The separability of a function can be destroyed through coordinate system rotation. This idea was applied to a reevaluation of the suitability of genetic algorithms for function optimization by Salomon (1996), who developed an algorithm for random rotations about multiple axes. This algorithm produces massively non-separable functions from separable ones. Salomon showed that standard genetic algorithms using low mutation rates in the order of a single bit-flip per chromosome are implicitly performing the relaxation method; and, that by destroying separability, the difficulty of the optimization task is increased significantly. We present a Lisp implementation of the Salomon coordinate rotation algorithm in appendix C and use it in our experiments below.

Although a determination of the actual likelihood of encountering problems as non-separable as these in the "real-world" is beyond the scope of this dissertation, our suspicion is that they are pathological. Nonetheless, they are important in exploring the robustness of our computational model of cooperative coevolution.

4.4.3 Test Suite

We now describe the four test functions we will use for determining the sensitivity of coevolutionary and standard evolutionary models to highly ordered epistatic interactions. All of these functions will be optimized with and without coordinate rotation, and have been defined such that their global minimums are zero. The first three functions were selected because their epistatic interactions form geometric lattices of a variety of sizes of peaks and basins on their fitness landscapes. All three of these functions are separable. The fourth is a non-separable function that was selected specifically because its interactions, although ordered, neither form a lattice nor are aligned with the coordinate system.

The first function in the test suite was originally proposed by Ackley (1987) and later generalized by Bäck and Schwefel (1993). It is defined as

$$f(\vec{x}) = -20\exp\left(-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}\right) - \exp\left(\frac{1}{n}\sum_{i=1}^{n}\cos\left(2\pi x_i\right)\right) + 20 + e,$$

where $n = 30$ and $-30.0 \leq x_i \leq 30.0$. The global minimum of zero is at the point $\vec{x} = (0, 0, \cdots)$. At a low resolution the landscape of this function is unimodal; however, the second exponential term covers the landscape with a lattice of many small peaks and basins.

The second test function is a generalized version of a function proposed by Rastrigin (1974). It is defined as

$$f(\vec{x}) = nA + \sum_{i=1}^{n} x_i^2 - A\cos(2\pi x_i),$$

where $n = 20$, $A = 3$, and $-5.12 \leq x_i \leq 5.12$. The global minimum of zero is at the point $\vec{x} = (0, 0, \cdots)$. The Rastrigin function is predominantly unimodal with an overlying lattice of moderate-sized peaks and basins.

The third function in the test suite was proposed by Schwefel (1981) and is defined as

$$f(\vec{x}) = 418.9829n + \sum_{i=1}^{n} x_i \sin\left(\sqrt{|x_i|}\right),$$

where $n = 10$ and $-500.0 \leq x_i \leq 500.0$. We have added the term $418.9829n$ to the function so its global minimum will be zero, regardless of dimensionality. The landscape of the Schwefel function is covered with a lattice of large peaks and basins. Its predominant characteristic is the presence of a second-best minimum far away from the global minimum—intended to trap optimization algorithms on a suboptimal peak. Unlike the previous two functions, the best minimums of this function are close to the corners of the space rather than centered. The global minimum occurs at the point $\vec{x} = (-420.9687, -420.9687, \cdots)$.

The final function in this test suite was proposed by Rosenbrock (1960) and was originally defined as

$$f(\vec{x}) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2,$$

where $-2.048 \leq x_i \leq 2.048$. The global minimum of zero is at the point $(1, 1)$. We will be optimizing an extended version of the Rosenbrock function proposed by Spedicato (1975) that is defined as

$$f(\vec{x}) = \sum_{i=1}^{n/2} \left[100(x_{2i} - x_{2i-1}^2)^2 + (1 - x_{2i-1})^2\right],$$

where $n = 20$. Unlike the other three functions in this test suite, the landscape of the Rosenbrock function is not covered by a lattice of peaks and basins. Rather, the function is characterized by an extremely deep valley whose floor forms a parabola $x_1^2 = x_2$ that leads to the global minimum. Given the nonlinear shape of the valley floor, a single rotation of the axes does not make the problem significantly easier, and the function should be relatively invariant to the type of coordinate system rotation we perform here. The function was designed by Rosenbrock to test his method of successive coordinate rotation that continuously tracks a curved ridge or valley.

For plots of two-dimensional versions of all four of these test functions, see appendix B.

4.4.4 Experimental Results

In all experiments, we initialize the populations of each species randomly, use a population size of 100, a two-point crossover rate of 0.6, a bit-flipping mutation rate set to the reciprocal of the chromosome length, fitness proportionate selection, and balanced linear scaling. Unless otherwise noted, fitness curves are generated from an average of 50 runs, and represent the function value produced from the best set of variable values found so far.

The graph in figure 4.12 on the following page shows the result of optimizing the Ackley function over a period of 500 generations with and without coordinate rotation. From the coevolution fitness curves, one can clearly see an initial period of slow adaptation as each of the species is successively evaluated within the context of a random individual from the remaining unevaluated species and the best individual from those already evaluated. The Ackley function being optimized here has 30 variables, and thus 30 species are coevolved; therefore, this initial phase lasts 30 generations. Once every species in the ecosystem has been evaluated at least one time, the optimization rate of cooperative coevolution dramatically increases and is far superior to the standard genetic algorithm when the epistatic interaction lattice is aligned with the coordinate system. However, the performance of the coevolutionary model is severely degraded by randomly rotating the coordinate system about multiple axes. A similar pattern is seen for the coevolutionary model when applied to the Rastrigin and Schwefel functions in figures 4.13 and 4.14.

The performance degradation of the coevolutionary model may be explained by its susceptibility to becoming frozen in Nash equilibrium. Recall that Nash equilibrium refers to a state in which all interacting species are optimal with respect to each other. This particular coevolutionary implementation is susceptible because it constrains the search space by adapting each species within the context of the current best individual from the others. When the function being optimized is separable—as the Ackley, Rastrigin, and

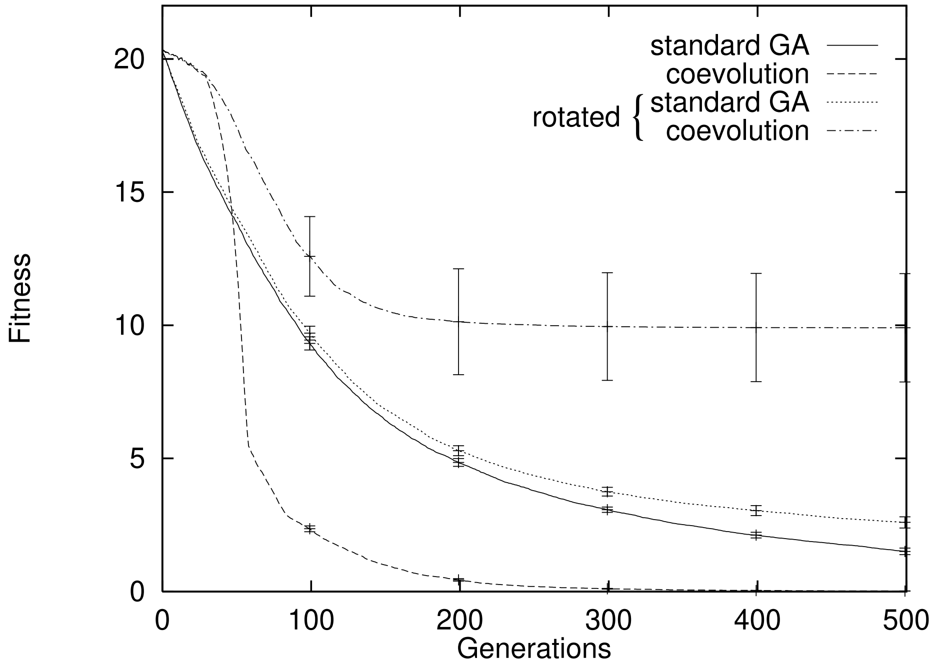


Figure 4.12: Sensitivity of coevolution and standard genetic algorithm to coordinate rotation of Ackley function

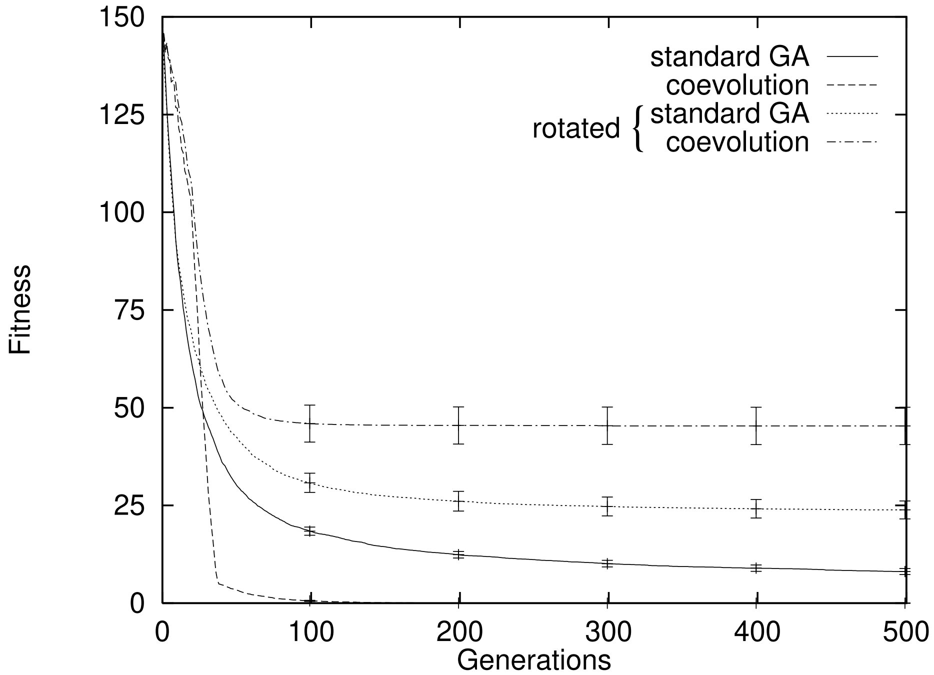


Figure 4.13: Sensitivity of coevolution and standard genetic algorithm to coordinate rotation of Rastrigin function

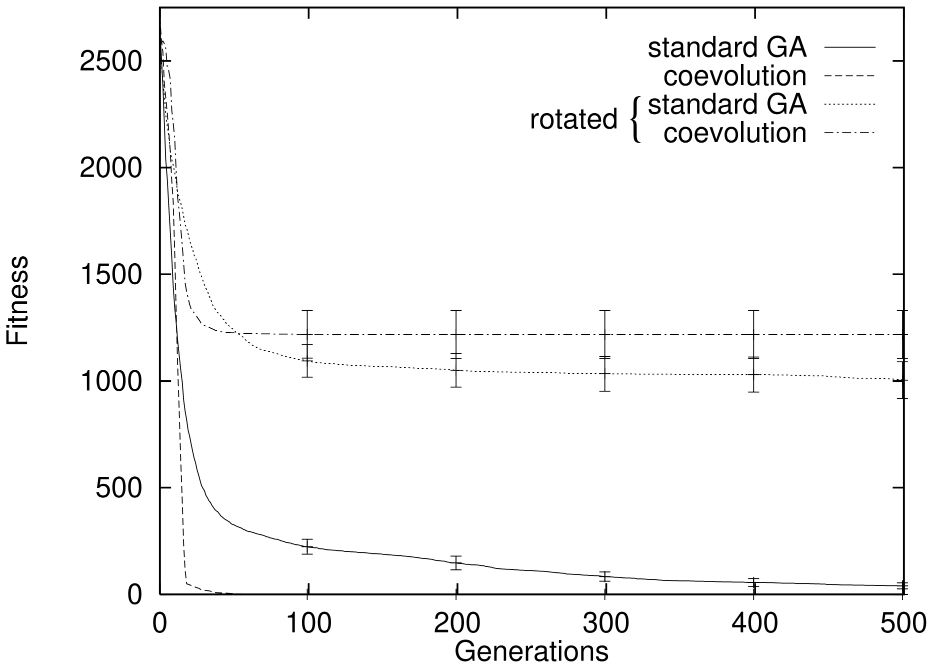


Figure 4.14: Sensitivity of coevolution and standard genetic algorithm to coordinate rotation of Schwefel function

Schwefel functions are—the only Nash equilibrium point is the global optimum. However, by rotating the coordinate system such that all variables are interdependent, we introduce a Nash equilibrium point at each local optimum, that is, at the bottom of each basin in the lattice. When only some of the variables are interdependent, as is the case in the experiment described here, each *independent* variable represents a path of escape. However, an optimization algorithm will again be susceptible to becoming frozen in Nash equilibrium when the independent variables have achieved their globally optimum values. Of course when all the dimensions are considered, this may still represent a point far from the true global optimum.

The standard evolutionary model is not as susceptible to Nash equilibria because multiple variables can be changed simultaneously. As a result, the performance degradation seen in this model due to coordinate system rotation takes a somewhat different form from that seen in the coevolutionary model. Specifically, the amount of degradation appears to be correlated with the size of the suboptimal basins covering the surface. Recall that the Ackley fitness landscape is dominated by a unimodal component and has an overlying lattice of small peaks and basins (see figure B.1 on page 145). Furthermore, the unimodal component is centered in the search space—making it rotation-invariant. The standard evolutionary model is able to adapt to this unimodal component without being excessively mislead by the lattice; therefore, there is not much degradation in performance due to coordinate system rotation. This can be clearly seen in figure 4.12 on the facing page. The Rastrigin function is similar; however, the sizes of the basins in its lattice are considerably larger relative to the unimodal component. One can see from figure 4.13 a corresponding increase in rotation-induced degradation. Unlike the first two functions, the Schwefel function has no unimodal component, and its fitness landscape is completely dominated by a lattice of large peaks and basins. As might be predicted, one sees a severe rotation-induced performance degradation in figure 4.14.

For completeness, one additional point needs to be made concerning the Schwefel function. Since its best minimums are located near the corners of the fitness landscape, rotation can easily move them outside the space under consideration. Since the coordinate rotations in this experiment are random, there will typically be a different post-rotation global optimum value for each run performed. The precise effect this has on the minimum post-rotation fitness shown in figure 4.14 has not been computed. However, based on the shape of the Schwefel landscape plotted in figure B.3 on page 147, we assume the effect is minimal. This is not an issue with the first two functions because their global optimum values are centered in the space and consequently are not affected by rotation.

Finally, the graph in figure 4.15 on the facing page shows the result of applying coevolution and the standard evolutionary model to the task of optimizing the Rosenbrock function. Recall that the Rosenbrock fitness landscape is not structured as a lattice. Instead, it is dominated by a deep valley whose floor forms a parabola that leads to the global minimum. Due to the curved shape of the valley, we speculated earlier that it should be relatively invariant to the type of coordinate system rotation we perform here. As expected, here we see much less of a rotation-induced performance degradation with the coevolutionary model than we saw in the previous three graphs. After about 100 generations, the post-rotation performance for coevolution is actually comparable to the pre-rotation performance for the standard genetic algorithm. This is an important result because it shows that it is not enough simply to consider variable interdependency when determining the suitability of an optimization method. One must also roughly classify the interdependency structure. We will return to the Rosenbrock function in section 4.5.

Alternative Collaboration Strategies

Given the susceptibility of the coevolutionary model to becoming frozen in Nash equilibrium due to its greedy collaboration strategy, the investigation of alternative methods for forming collaborations between species is clearly an important topic for future research. Although we defer a detailed study of this topic, we show in figure 4.16 on the next page the result of optimizing the Ackley function with a slightly less greedy strategy. When using the alternative strategy, there is a dramatic decrease in susceptibility to Nash equilibria.

Specifically, the less greedy interaction strategy is to evaluate each individual within the context of two collaborations. The first collaboration is formed as in the greedy method; that is, a vector of variable values is constructed that consists of the value represented by the individual being evaluated and the current best variable value from each of the other species. The second collaboration is constructed from the individual being evaluated and *random* individuals from each of the other species. Both vectors are then applied to the objective function, and the better of the two results is taken to be the fitness of the individual being evaluated. Our claim is not that this less greedy method is the "right" collaboration strategy to use, but rather that there is evidence to warrant further study of alternative methods. We will revisit this topic in chapter 7 when discussing directions for future research.

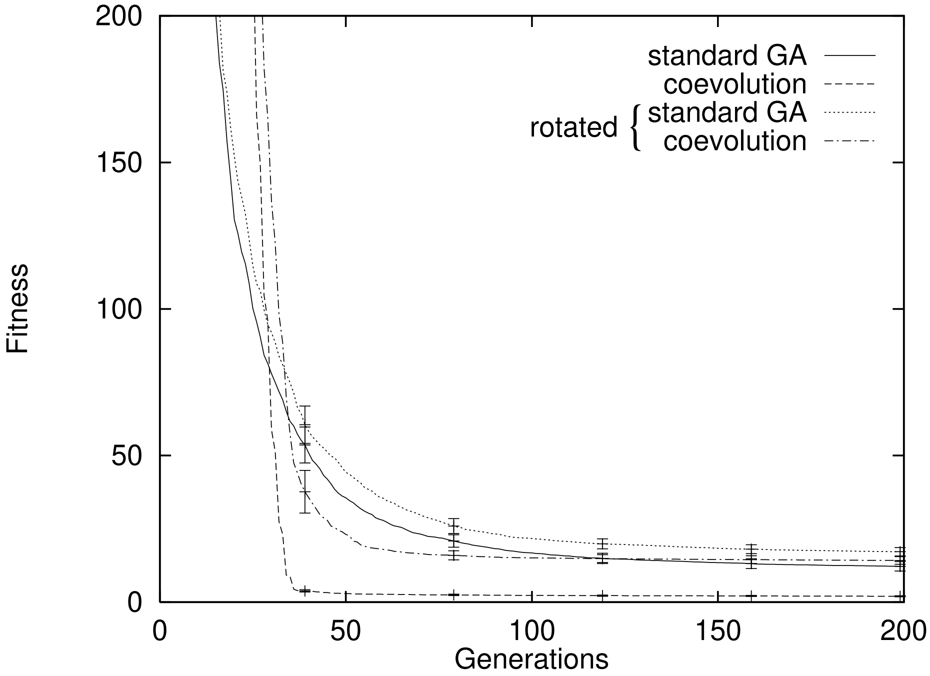


Figure 4.15: Sensitivity of coevolution and standard genetic algorithm to coordinate rotation of extended Rosenbrock function

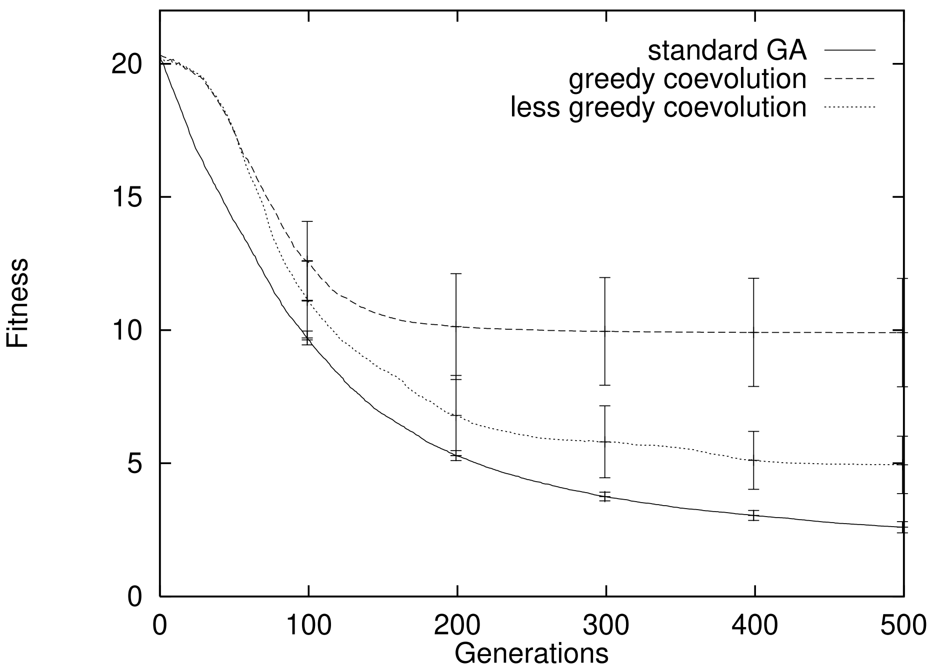


Figure 4.16: Effect of a less greedy collaboration strategy on the optimization of the rotated Ackley function

4.5 Sensitivity to Dimensionality

When faced with a problem of only a few variables we can often find the optimum simply by taking partial derivatives and solving the resulting system of equations. However, as the dimensionality of the problem increases, this becomes difficult, if not impossible, even with the most powerful computers. As a result, we often must resort to the use of heuristic methods, of which evolutionary algorithms are an example, that are not guaranteed to find the global optimum. In his classic book on dynamic programming, mathematician Richard Bellman (1957) refers to the difficulties associated with optimizing functions of many variables as the "curse of dimensionality".

This "curse" is not restricted to the domain of function optimization. When we build computational models of biological systems or explore the domain of artificial intelligence we often experiment within the confines of an extremely simplified universe; see, for example, (Thrun et al. 1991). Clearly, it is important to be concerned with the effect of an increase in scale on these models and techniques. In the domain of artificial intelligence, a number of researchers have been focusing on the issue of scalability. Three examples include the work of Doorenbos (1994) in the area of rule-based systems, who has successfully improved the Rete match algorithm[5] to handle large collections of over 100,000 rules with little degradation in performance; the work of Lenat (1995) in the area of knowledge representation, who for over a decade has been working on a system called CYC that currently includes a database of several million "commonsense axioms"; and the work of de Garis (1996) in the area of artificial neural networks, whose stated goal for the year 2001 is the construction of "artificial brains with a billion neurons".

In this section, we study the effect of increasing dimensionality on cooperative coevolution. We briefly discussed the advantages of the model with respect to parallelism in section 3.2.5. However, the facility for constructing a highly parallel coevolutionary model of a problem only partially addresses scalability. One must also be concerned with the increase in complexity of interaction among subcomponents that occurs as a result of an increase in scale. We defer a detailed study of a parallel implementation of the model and concentrate on the complexity issue here.

4.5.1 Test Suite

We will use two test functions in our experiments for determining the sensitivity of the coevolutionary and standard evolutionary models to an increase in dimensionality. As in the previous section, these functions have been defined such that their global minimums are zero. One of the primary considerations in selecting the two functions was that they represent the class of separable and non-separable functions respectively.

The first function in the test suite is the sphere model—a very simple quadratic with hyperspherical contours defined as

$$f(\vec{x}) = \sum_{i=1}^{n} x_i^2,$$

[5]The Rete match algorithm is designed to perform efficient pattern matching through the use of a discrimination network called a Rete net (Forgy 1982).

where $-5.12 \leq x_i \leq 5.12$. We vary the dimensionality of this function from 10 to 80. The global minimum of zero is at the point $\vec{x} = (0, 0, \cdots)$. This function has been used previously in the development of evolution strategy theory (Rechenberg 1973), and in the evaluation of genetic algorithms as part of the De Jong test suite (De Jong 1975). It exemplifies the class of separable functions that have proven to be easily optimized by evolutionary computation. The second function in this test suite is the extended Rosenbrock function previously described in section 4.4.3. As with the sphere model, we vary its dimensionality from 10 to 80. Two-dimensional versions of both of these functions are plotted in appendix B.

4.5.2 Experimental Results

As in the study on highly ordered epistatic interactions, in all experiments we initialize the populations of each species randomly, use a population size of 100, a two-point crossover rate of 0.6, a bit-flipping mutation rate set to the reciprocal of the chromosome length, fitness proportionate selection, and balanced linear scaling. All fitness curves are generated from an average of 50 runs and represent the function value produced from the best set of variable values found so far.

The graphs in figure 4.17 on the following page show the result of varying the dimensionality of the sphere model. The top graph was generated from optimization runs on functions of 10 and 20 variables, and the bottom graph was generated from functions of 40 and 80 variables. Although the graphs have different scales for clarity, a 1:1 aspect ratio is maintained in both to facilitate a direct comparison. The graphs show the characteristic gradually increasing rate of adaptation by the coevolutionary model as the species are initially evaluated. This is followed by a period of rapid adaptation in which the coevolutionary model performance far surpasses the performance of the standard evolutionary model. Although the optimization task clearly becomes more difficult as the dimensionality of the problem increases, the relative performance of the two models on this separable function appears not to be significantly affected.

Next, the same experiment is performed with the non-separable extended Rosenbrock function. In the extended Rosenbrock function, each variable has a symmetric dependency link with one other. Specifically, variables x_1 and x_2 are interdependent; variables x_3 and x_4 are interdependent; and so on. Therefore, when a Rosenbrock function of n variables is optimized with an n-species coevolutionary model, there will be n epistatic interactions in the ecosystem. In other words, the epistatic interactions increase linearly with dimensionality. Instead of this relationship reducing the efficiency of coevolution when dimensionality is increased as one might expect, figure 4.18 on page 67 shows that there is a corresponding *increase* in the performance of the coevolutionary model relative to the standard evolutionary model. When the low communication overhead associated with evolving species on multiple processors is taken into account, these graphs provide strong evidence that the coevolutionary model could be effectively applied to extremely large problems.

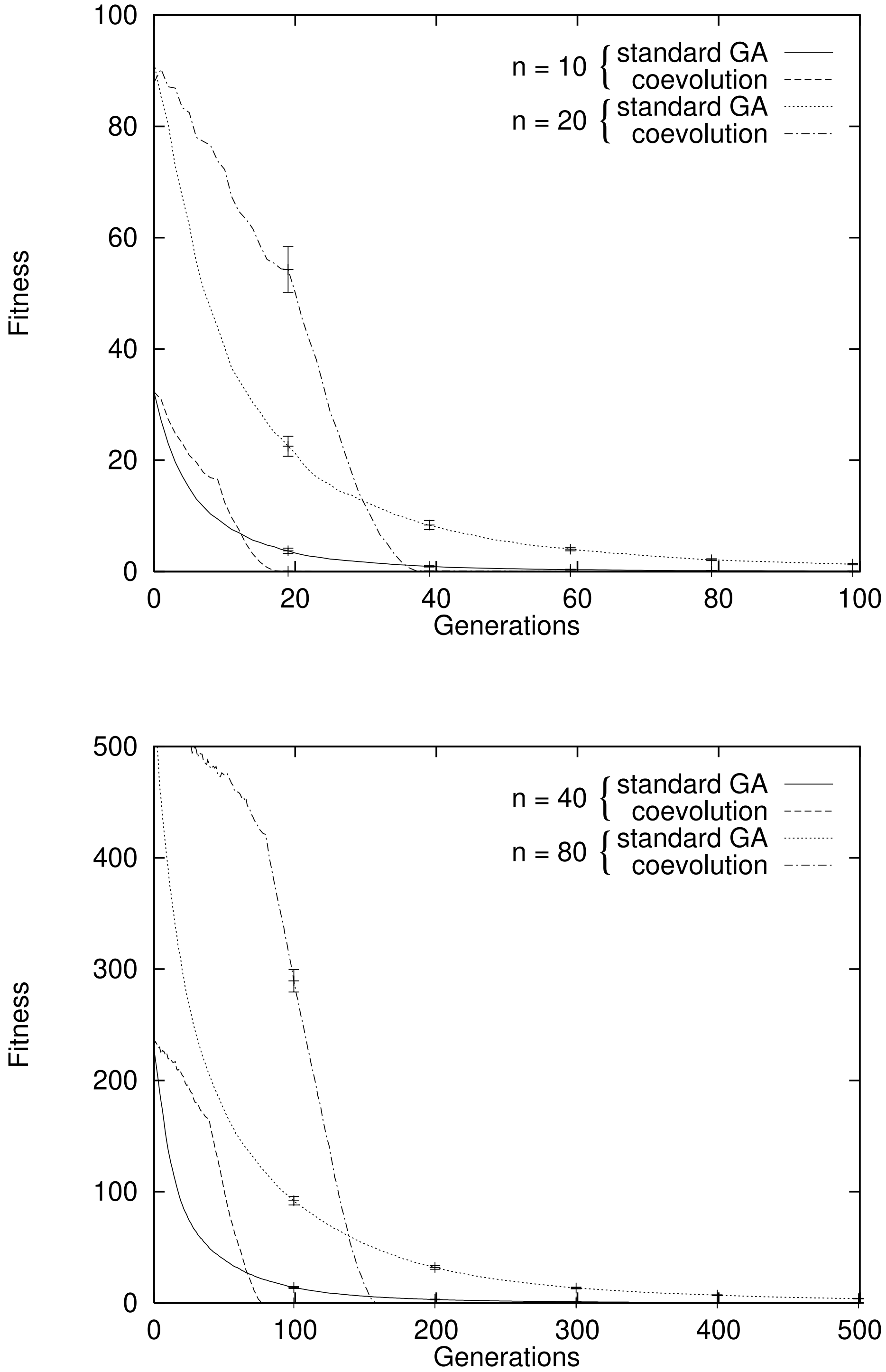
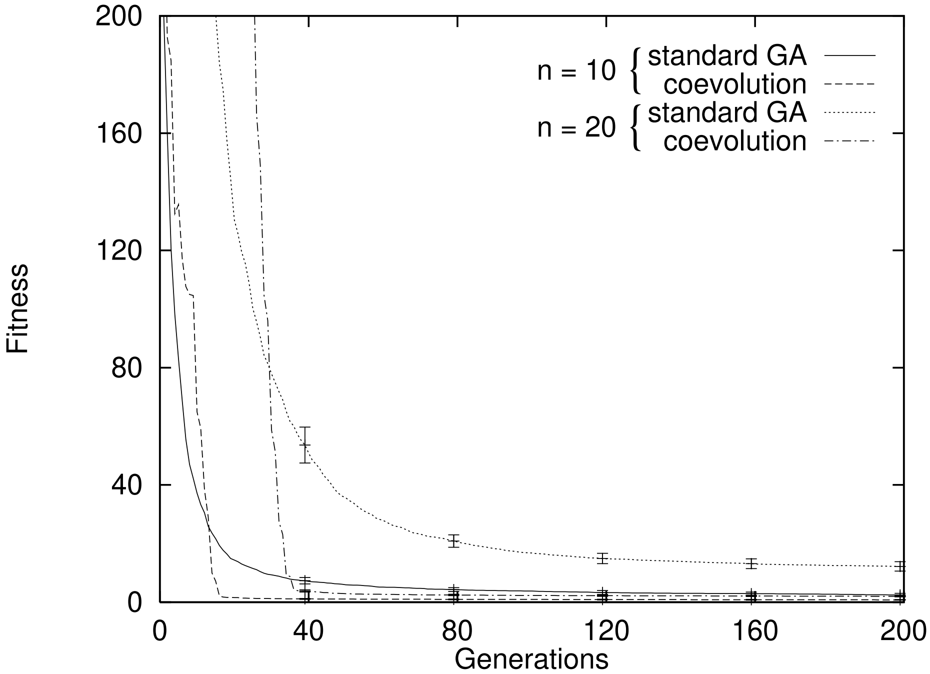


Figure 4.17: Sensitivity of coevolution and standard genetic algorithm to changes in dimensionality of sphere model



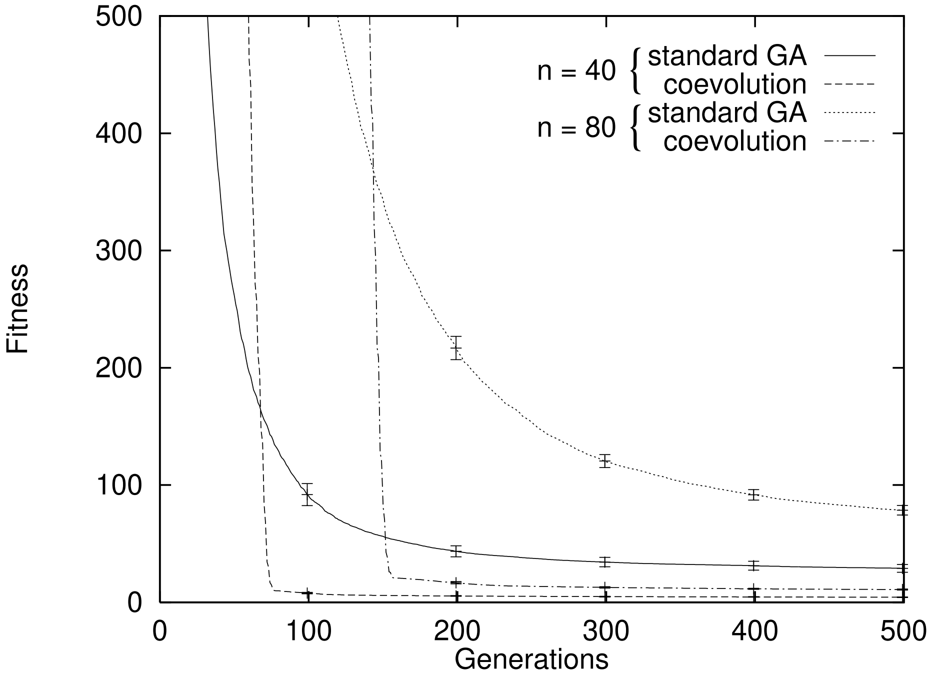


Figure 4.18: Sensitivity of coevolution and standard genetic algorithm to changes in dimensionality of extended Rosenbrock function

4.6 Sensitivity to Noise

Evolutionary algorithms have been shown to be somewhat resistant to the effect of noise in the fitness evaluation procedure. As a result, when the fitness evaluation is computationally expensive it may be better to do less accurate evaluations because this will enable one to adapt the population over a greater number of generations (Grefenstette and Fitzpatrick 1985; Fitzpatrick and Grefenstette 1988). The primary negative effect of noise on the standard evolutionary model is the misallocation of reproductive cycles to weak members of the population. Our hypothesis is that this effect will be compounded in the coevolutionary model because it may lead to a poor choice of species representatives, which in turn will distort the evaluation of all the collaborations the representative participates in. This suggests that the coevolutionary model may be less suitable for problems with noisy objective functions.

In this section we investigate the effect of noise by applying both coevolution and the standard evolutionary model to the task of function optimization in an environment where the amount of fitness evaluation noise can be varied.

4.6.1 Test Suite

As in the dimensionality study, we use separable and non-separable test functions in this sensitivity analysis. The test functions are defined such that their global minimums are zero.

The separable stochastic function in this test suite was proposed by De Jong (1975) for the performance evaluation of "genetic adaptive plans". The function is a high-dimensional unimodal quadratic with Gaussian noise defined as

$$f(\vec{x}) = \sum_{i=1}^{n} i x_i^4 + \mathrm{Gauss}(0, \sigma),$$

where $n = 30$ and $-1.28 \leq x_i \leq 1.28$. We vary the standard deviation of the Gaussian distribution from 1.0 to 8.0. With the noise filtered out, the global minimum of zero is at the point $\vec{x} = (0, 0, \cdots)$. For plots of this function with and without noise, see appendix B.

The non-separable stochastic function in this test suite is the extended Rosenbrock function described in section 4.4.3 with an additional Gaussian noise component defined as

$$f(\vec{x}) = \sum_{i=1}^{n/2} \left[100(x_{2i} - x_{2i-1}^2)^2 + (1 - x_{2i-1})^2\right] + \mathrm{Gauss}(0, \sigma),$$

where $n = 30$ and $-2.048 \leq x_i \leq 2.048$. As with the stochastic De Jong function, we vary the standard deviation of the Gaussian distribution from 1.0 to 8.0.

4.6.2 Experimental Results

As before, we initialize the populations of each species randomly, use a population size of 100, a two-point crossover rate of 0.6, a bit-flipping mutation rate set to the reciprocal of the chromosome length, fitness proportionate selection, and balanced linear scaling. All

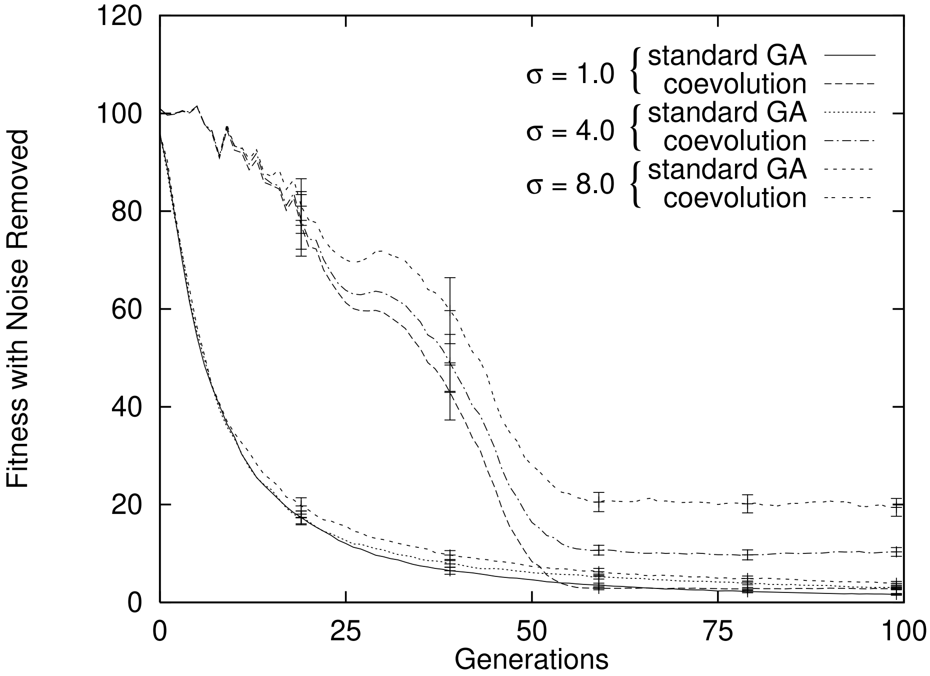


Figure 4.19: Sensitivity of coevolution and standard genetic algorithm to changes in the standard deviation of noise in stochastic De Jong function

fitness curves are generated from an average of 50 runs and represent the noise-free function value produced from the best set of variable values found so far. Although the noise has been filtered out of the fitness curves to more accurately show the proximity of the evolved solutions to the true optimum, these filtered fitness values were not available to the two evolutionary systems; that is, only the noisy evaluations were used for the allocation of trials.

The graph in figure 4.19 shows the result of varying the standard deviation of the noise in the stochastic De Jong function. The experiment was run with standard deviations of 1.0, 4.0, and 8.0. Although the coevolutionary model is able to handle a low level of noise, a degradation in its performance can clearly be seen in the graph as the noise is increased. As predicted, a comparison with the fitness curves generated by the standard genetic algorithm reveals that this particular coevolutionary implementation has more difficulty than the standard model when optimizing objective functions with a high level of noise. High levels of noise in the non-separable stochastic function produced a similar pattern of performance degradation as shown in figure 4.20 on the next page.

It is encouraging that the coevolutionary model is able to handle low levels of noise in the fitness evaluation of collaborations without difficulty. Although overcoming higher levels of noise as in the experiments described here may require multiple fitness evaluations to reduce the amount of uncertainty, we do not consider this a major problem. Furthermore, the difficulties are a reflection of our particular implementation; not of coevolution in general. Almost certainly, alternative collaboration strategies could be found that have a higher resistance to the negative effect of noisy fitness evaluations.

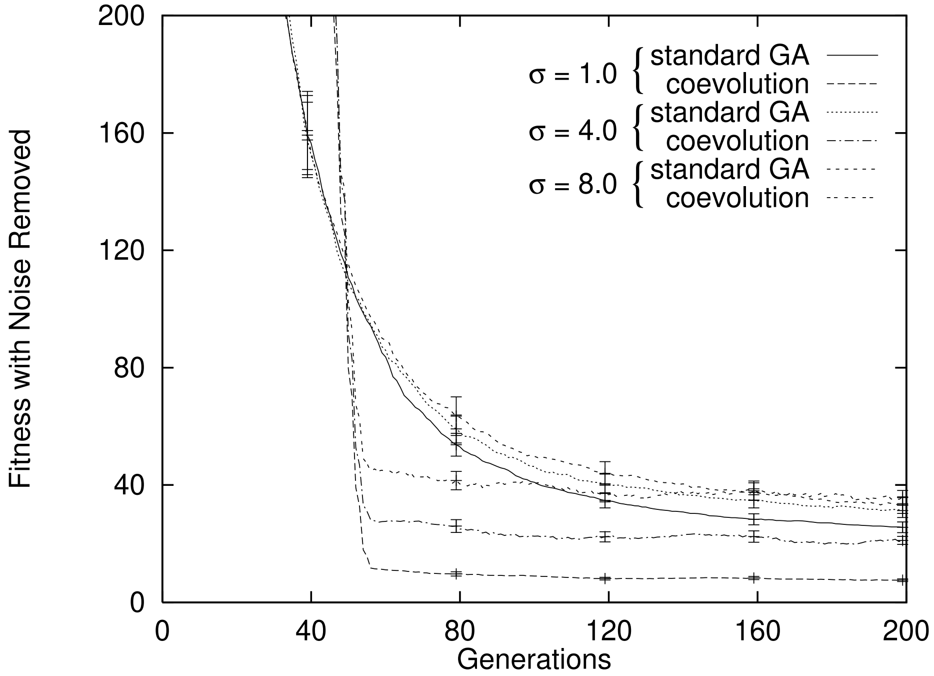


Figure 4.20: Sensitivity of coevolution and standard genetic algorithm to changes in the standard deviation of noise in stochastic Rosenbrock function

4.7 Summary

In summary, this chapter performed a sensitivity analysis on a number of characteristics of decomposable problems likely to have an impact on the performance of the coevolutionary model. The characteristics include the amount and structure of interdependency between problem subcomponents, which we characterized as epistatic interactions; the dimensionality or scale of the problem; and the amount of noise in the fitness evaluations. Our goal was to gain insight into the circumstances under which these characteristics will have a negative impact on the performance of the model and how any exposed difficulties may be overcome. This was accomplished using tunable test functions chosen specifically to measure the effect of the target characteristics.

In section 4.3 we described Kauffman's tunable NK model of fitness landscapes, which is designed to capture the statistical structure of the rugged multipeaked fitness landscapes seen in nature. We demonstrated that as the level of epistatic interactions between genes within a chromosome increases, thus decreasing the correlation between genotype Hamming distance and fitness, it becomes increasingly difficult for evolutionary computation to find highly fit points on the landscape. However, there is a corresponding increase in the performance of cooperative coevolution relative to the performance of the standard evolutionary model. What is more important, we showed that an increase in the random epistatic interactions between species has little effect on the relative performance of the two models. This has positive implications for the applicability of cooperative coevolution to the solution of a broad class of problems with complex interdependencies between subcomponents.

Next, in section 4.4 we characterized the structure of epistatic interactions that often occur in the domain of real-valued function optimization as highly ordered. Closely related to the structure of epistatic interactions is the notion of function separability. Many problems from the domain of function optimization are separable; and, due to biases in both our coevolutionary model and the standard evolutionary model, separable functions can be effectively optimized with these methods. However, by making the functions massively non-separable through coordinate system rotation, the difficulty of the optimization task is increased significantly. When the structure of the interactions forms a geometrical lattice, rotation negatively affects the coevolutionary model more than the standard evolutionary model. This is likely due to the greedy collaboration strategy used, which makes the coevolutionary model highly susceptible to becoming frozen in Nash equilibrium. However, rotation degrades the performance of the coevolutionary model much less when the interactions produce a landscape of curved ridges and valleys rather than a lattice of peaks and depressions. We suggested that further research into alternative strategies for forming collaborations is warranted.

In section 4.5 the scalability of the coevolutionary model was investigated. Experiments were performed on an easy separable function and on a more difficult non-separable function. While all the separable function variables were independent, the number of interdependencies between variables in the non-separable function increased at the same rate as the number of variables. Although an increase in dimensionality of the separable function had little effect on the relative optimization performance of the coevolutionary and standard evolutionary models, surprisingly, the performance of coevolution increased relative to the standard model when the dimensionality of the non-separable function was increased. This somewhat counter-intuitive but encouraging result suggests that coevolution may be suitable for the solution of extremely large problems; especially when one considers the potential for parallelizing the model.

In the final section, the sensitivity of coevolution to noise in the evaluation function was investigated. Again, experiments were performed on both a separable and a non-separable function. In optimizing both functions, the coevolutionary model was resistant to low levels of noise. However, its performance degraded faster than the standard evolutionary model as the level of noise was increased. The negative effect of noise on coevolution appears to be compounded. First, there is an initial misallocation of reproductive cycles to weak members of the population. This in turn can lead to a poor choice of species representatives, which distorts the evaluation of all the collaborations in which the representatives participate. We emphasize that this is a reflection of our particular implementation; not of coevolution in general. Almost certainly, alternative collaboration strategies could be found that have a higher resistance to the negative effect of noisy fitness evaluations.

Chapter 5

BASIC DECOMPOSITION CAPABILITY OF THE MODEL

One of the primary issues that must be addressed if a complex problem is to be solved through the evolution of coadapted subcomponents is decomposition; that is, how to determine an appropriate number of subcomponents and the precise role each will play. Earlier in this dissertation we made a number of claims concerning the ability of cooperative coevolution to address adequately the issue of problem decomposition. However, up to this point all of our experiments have involved a static hand-decomposition of the problem. In this chapter we will show that problem decomposition is an emergent property of cooperative coevolution.

Many task-specific methods exist for finding good problem decompositions. In chapter 2 we described a number of problem decomposition methods that have been previously used in the field of evolutionary computation. For example, classifier systems utilize a complex bidding mechanism based on a micro-economy model to decompose the problem into a collection of coadapted rules, while other evolutionary systems simply hand-decompose the problem. Examples of task-specific methods that have been used in non-evolutionary systems include statistical approaches and techniques utilizing symbolic logic.

In contrast to these earlier problem decomposition techniques, cooperative coevolution takes a task-independent approach in which the decomposition emerges purely as a result of evolutionary pressure. That is, good decompositions have a selective advantage over poor decompositions. The goal of this chapter is to take an initial step in determining whether evolutionary pressure alone is sufficient for producing good decompositions by exploring the basic capability of the model to perform this task.

In the following empirical analysis we will describe four experiments—each designed to answer a specific question concerning the ability of cooperative coevolution to decompose problems. The questions are as follows:

- Will a collection of species locate multiple environmental niches and work together to cover them?
- Will each species evolve to an appropriate level of generality?
- Will species adapt to changes in the environment?
- Will the occasional creation of new species and the elimination of unproductive ones induce the emergence of an appropriate number of species?

5.1 String Covering Problem

To provide a relatively simple environment in which the emergent decomposition properties of cooperative coevolution can be studied, we return to the string covering problem first described in example 2.4 beginning on page 16. Recall that the goal is to evolve a set of binary strings that matches a set of target strings as closely as possible. The match strength between two strings is computed by summing the number of bits in the same position with the same value. We refer to the set of evolving strings as the *match set* and the target strings as the *target set*. The fitness of a particular match set is computed by averaging the maximum match strengths produced for each target string as described in equation 3.3 on page 40.

Along with providing a common framework for the four experiments comprising this empirical analysis, string covering is an important application in its own right. One reason for its importance is that it can be used as the underlying mechanism for modeling a number of complex processes from nature, for example, the discrimination between self and non-self that occurs within the vertebrate immune system. We will explore the connection between string covering and the immune system in detail in chapter 6.

5.2 Evolving String Covers

Each of the four experiments comprising this analysis will involve applying a cooperative coevolutionary genetic algorithm, similar to the implementation described in at the end of chapter 3, to the string covering problem just described. Each species in the ecosystem contributes a single string to the match set. In evaluating the individuals from one species, each will collaborate with the current best individual from each of the other species in the ecosystem. In other words, a match set will consist of a single individual from the species being evaluated, and the current best individual from each of the other $N-1$ species. Since the individuals being evolved are binary strings, no distinction needs to be made between their genotypes and phenotypes.

In all experiments, we initialize the populations of each of the species randomly, use a population size of 50, a two-point crossover rate of 0.6, a bit-flipping mutation rate set to the reciprocal of the chromosome length, fitness proportionate selection, and balanced linear scaling.

5.3 Locating and Covering Multiple Environmental Niches

The first question we seek to answer is,

> Will a collection of species locate multiple environmental niches and work together to cover them?

In a previous study by Forrest et al. (1993), an experiment was performed demonstrating the ability of a single-population genetic algorithm using emergent fitness sharing to detect common schemata in a large collection of target strings. By *schemata* we are referring to string templates consisting of a fixed binary part and a variable part often designated by the symbol '#'. To compute the match strength between two strings, they used the simple linear function we previously described in equation 3.2 on page 40. The Forrest schema detection experiment is duplicated here, substituting cooperative coevolution for the genetic algorithm used in their study.

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.
```

In other words, 100 of the strings in the target set will begin with a sequence of 32 ones and the other 100 strings will end with a sequence of 32 ones. 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.
```

Note that 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. This is because the fixed regions that define the niches of the eighth-length schemata are smaller with respect to the variable region of the strings.

Since we know *a priori* how many niches exist and are only interested in whether we can locate and cover them, we simply evolve an equal number of species as niches. We defer to section 5.6 the issue of determining an appropriate number of species when this information is not available beforehand. 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.

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 5.1 on the next page. 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

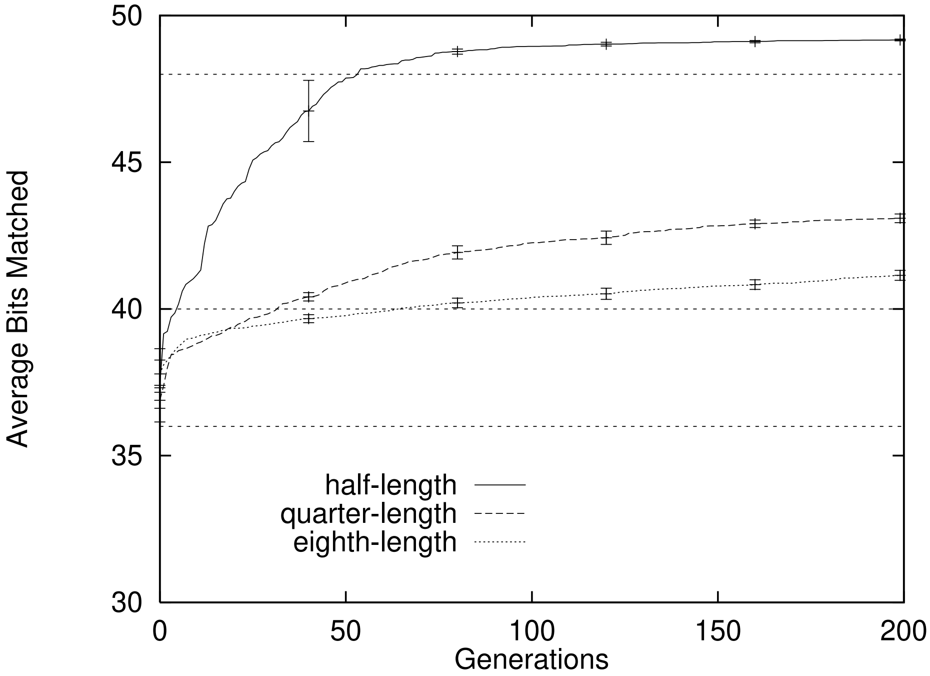


Figure 5.1: Finding half-length, quarter-length, and eighth-length schemata

intervals. The dashed 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 standard genetic algorithm consistently evolves this best possible single-string generalist. Figure 5.1 shows that when multiple species collaborate, they are able to cover the target set better than any single individual evolved with a standard genetic algorithm. Furthermore, when more species are employed, as in the eighth-length schema experiment, the amount of improvement over a standard genetic algorithm increases.

The reason for this improvement can be seen in figure 5.2 on the following page. 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. The substrings perfectly matching the fixed regions of the schemata have been highlighted in each individual for ease of viewing. A couple of observations can be made from this figure. First, it is clear that each species focuses on one or two target string subsets and relies on the other species to cover the remaining target strings. This enables the species to cover their respective subsets 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, occasionally two or more species may occupy a common niche, 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

Half-length

Species 1: 1111111111111111111111111111111110100110001000111111100010110111
Species 2: 0010000001001110110100001000100011111111111111111111111111111111

Quarter-length

Species 1: 1111111111111111000110011001010011111111111111110111111111110111
Species 2: 1111111110111111111111101011111111111111111111111000010100010000
Species 3: 0101010101101101111111011111110100001010001011101111111111111111
Species 4: 1001101110011000111111111111111110110100010100011010101101101111

Eighth-length

Species 1: 1100100011111100010111101000000011001011111111110011010011110010
Species 2: 1011111001010001111111110010101110010000101101111110111011010010
Species 3: 1010111100000111111101111001000011100110011110111101111100000111
Species 4: 0000111011111111101110010111101011111111010001101100010111111101
Species 5: 1101100100100010110000111100101111111111100101011001000111111111
Species 6: 0001011011111011011010001111111100110011111111101111111100110000
Species 7: 11110010111111101000000101011010100100100100100010111000111111111
Species 8: 1111111110001101111000001111111101011010110111101100001111101010

Figure 5.2: Final species representatives from schemata experiments

considered viable. Third, some of the species make no obvious contribution, for example, the third species from the eighth-length schemata experiment. 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. One such mechanism for removing unproductive species will be explored in section 5.6.

5.4 Finding an Appropriate Level of Generality

The next issue we will explore is,

> Will each species evolve to an appropriate level of generality?

This is an important question because, in solving covering problems, the covering set will typically have fewer elements than the number of items that need to be covered. Therefore, generalization to some degree is required.

To determine whether species will evolve to an appropriate level of generality, we performed a number of experiments using the following 32-bit test patterns[1]:

```
11111111111111111111111111111111
11111111110000000000000000000000
00000000000000000000001111111111.
```

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

```
11111111110000000000001111111111,
```

which produces an average match score of $(20 + 22 + 22)/3 = 21.33$. The best 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: $(32{+}20{+}24)/3 = 25.33$. Note that the makeup of the extreme left and right 10-bit segments of the second string is unimportant. Of course, the best three-string cover of the three patterns is simply the patterns themselves.

To build on the previous section in which it was shown that a collection of species can discover important environmental niches, we hid the three 32-bit test patterns by embedding them in three schemata of length 64. A target set composed of 30 strings was then generated in equal proportion from the schemata. The schemata are as follows:

```
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######.
```

Four experiments consisting of five runs each were performed. In the first experiment, we evolved a cover for the target set using a single species. This is of course equivalent to using a standard genetic algorithm. The remaining three experiments include evolving two, three, and four species respectively. The plots in figures 5.3, 5.4, 5.5, and 5.6 beginning on the next page show the number of target bits matched by the best individual from each species. They were generated from the first run of each experiment rather than the average of the five runs so that the instability that occurs during the early generations is not masked. However, we verified for each experiment that all five runs produced similar results. Although the figures show just the number of bits matching the 32-bit target patterns, the fitness of individuals was based on how well the entire 64 bits of each target string was matched.

In figure 5.3 we see that after an initial period of instability, the single 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 best single-string generalist described previously. Figure 5.4 shows that when we more fully utilize the model of cooperative coevolution by evolving two species, the first pattern

[1]Using 66-bit complements of the three test patterns shown, Forrest et al. (1993) experimented with the generalization capability of a single-population evolutionary algorithm utilizing emergent fitness sharing.

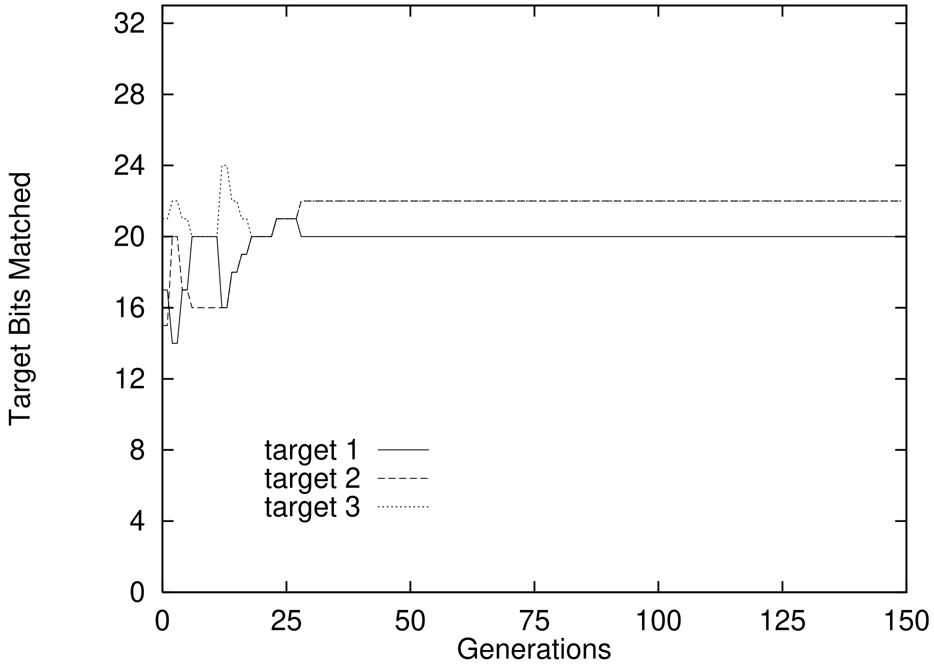


Figure 5.3: One species covering three hidden niches

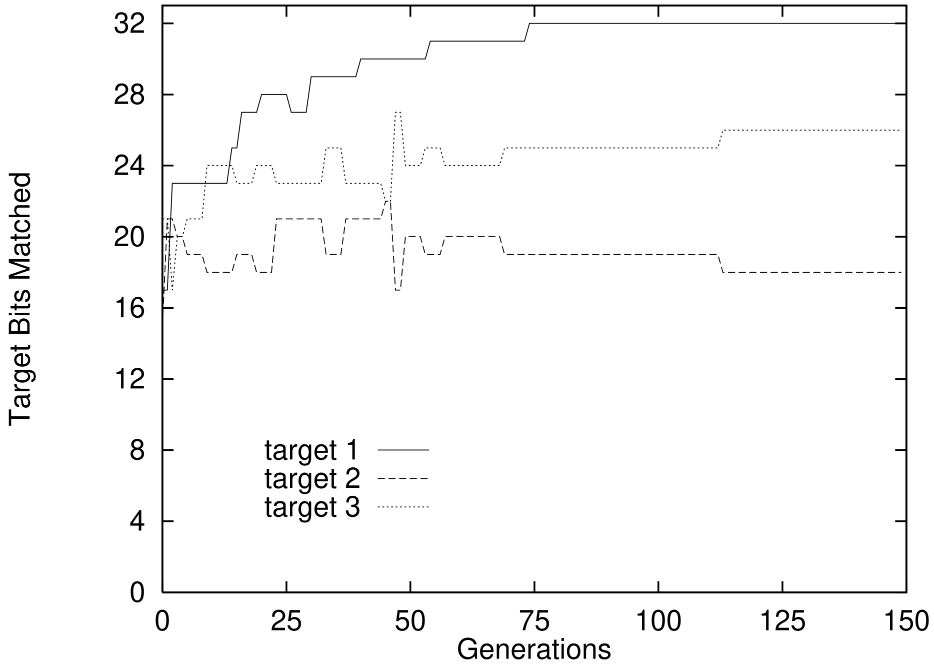


Figure 5.4: Two species covering three hidden niches

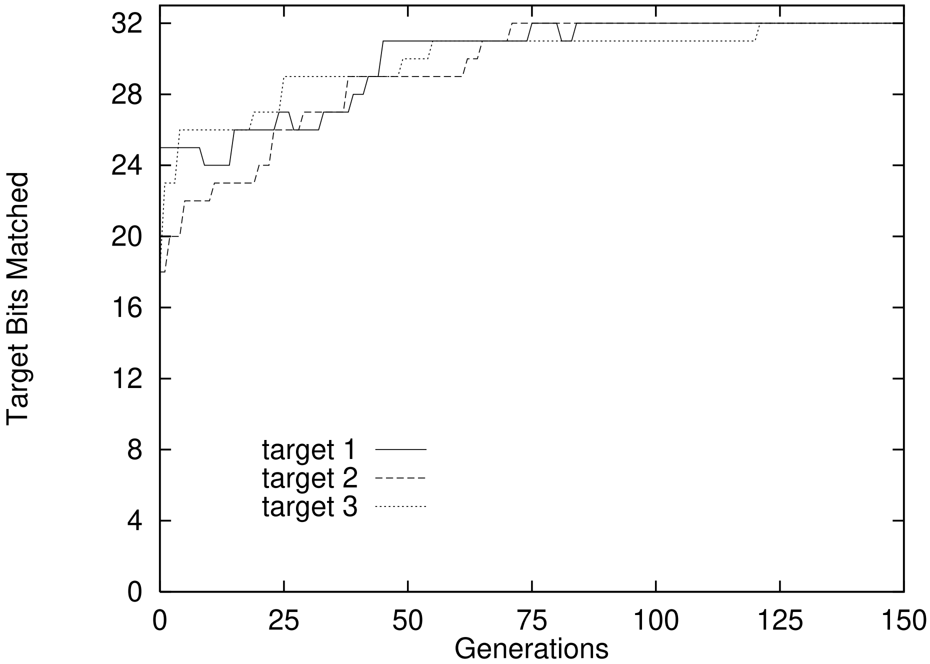


Figure 5.5: Three species covering three hidden niches

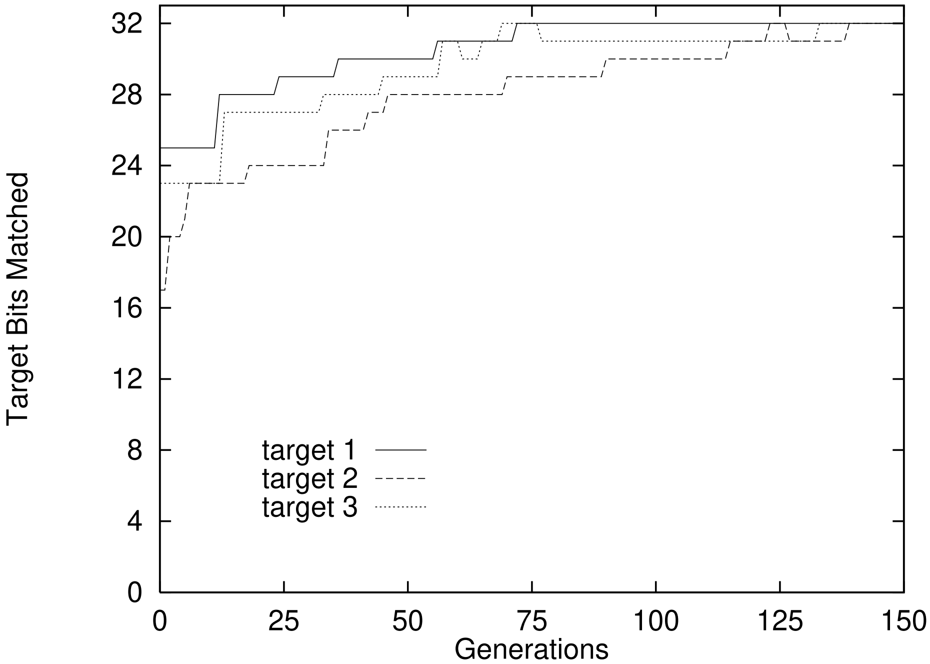


Figure 5.6: Four species covering three hidden niches

is matched perfectly and the other two patterns are matched at the level of 26 and 18 bits respectively—consistent with the best possible two-string generalization. When we increase the number of species evolved to three, all three patterns are matched perfectly at the level of 32 bits, as shown in figure 5.5. This indicates that each species has specialized on a different test pattern. Finally, in figure 5.6 we see that when the number of species is increased to four, perfect matches are also achieved. However, in terms of the number of fitness evaluations, this experiment required more resources to achieve perfect matches. Note that each generation plotted in all these figures represents 50 fitness evaluations.

Conclusive evidence that the species evolve to appropriate levels of generality can be seen in figure 5.7 on the facing page, which shows the best individual from each of the species at the end of the final generation. It also shows that after removing all the bits corresponding to the variable regions of the target strings, the patterns that remain are the best possible one, two, and three element covers described earlier.

One final observation from figure 5.7 is that by removing the bits corresponding to variable target regions in the four-species experiment, it becomes obvious 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 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. A p-value of 0.0000 produced by a two-sided t-test verifies that this difference is unlikely to have occurred by chance.

5.5 Adapting to a Dynamic Environment

Now that we know that the model is able to discover multiple environmental niches and evolve subcomponents appropriate in generality to cover them, the next question we need to answer is,

> Will species adapt to changes in the environment?

This question is important for a number of reasons. First, some of the tasks to which we may want to apply evolutionary algorithms will have non-stationary objective functions (Pettit and Swigger 1983; Goldberg and Smith 1987; Cobb 1990; Grefenstette 1992). Second, in a coevolutionary model, an adaptation by one species can change the objective functions of all the species it interacts with. Third, adding new species to the ecosystem is a major source of environmental change that will occur in any model of coevolution among interdependent species if we do not know *a priori* how many species are required, which will generally be the case, and must dynamically create them as evolution progresses. One of the results of incrementally creating species in our cooperative model is that as the new species begin making contributions, the older species are free to become more specialized.

We can observe many examples of non-stationary objective functions in nature. In natural ecosystems, fitness landscapes may change due to random processes such as climatic

One-species experiment

Species 1: 1001010100011111001001000101101010100001100111101111101011100110

Noise removed

1111111111000000000001111111111

Two-species experiment

Species 1: 1011000100011111001001111111111011011111111111101111101111110110

Species 2: 0100110000000110000010000101101010100000000100101111010011000110

Noise removed

1111111111111111111111111111111

0000011000000000000000001001110011

Three-species experiment

Species 1: 1001000100011111001001111111111111111111111111111111101111111100

Species 2: 01000100000000000000010000001001110100000100111101101101011000010

Species 3: 1001100100011111011001000101101000100000100000001010010100110010

Noise removed

1111111111111111111111111111111

0000000000000000000001111111111

1111111111000000000000000000000

Four-species experiment

Species 1: 1001000100011111001001111111111111111111111111111111101111111100

Species 2: 01000100000000000000010000001001110100000100111101101101011000010

Species 3: 11010111011111111101001000101101000100001100000001000000100101100

Species 4: 10110001010111110111010001010000000000001000000000010000100011000

Noise removed

1111111111111111111111111111111

0000000000000000000001111111111

1111111111000000000000000000000

1111111111000000000000000000000

Figure 5.7: Final representatives from one through four species experiments before and after the removal bits corresponding to variable target regions

changes. When this occurs species must adapt or be destroyed. For example, when a major drought occurred on Daphne Major Island in the Galápagos in 1977, the population of finches was dramatically reduced while the size of the surviving birds increased. The increase in the proportion of larger birds was presumably an adaptation to a change in their food source—mostly larger and harder to crack seeds were available during the drought (Boag and Grant 1981). Of course, the activity of a species may also have a major impact on its ecosystem, which in turn may warp the objective function of other species and force their adaptation. A classic example is melanism in the moth, *Biston betularia*, which occurred in industrial areas of Britain beginning around the year 1850 (Kettlewell 1955). Prior to 1850, all the moths were whitish gray with black speckles. This made the moths difficult to see when resting on the lichen covered tree bark of the region. However, as air pollution from industry began to kill the lichen, a black form of the moth called *carbonaria* appeared. The *carbonaria* blended in much better to darker tree surfaces. By the middle of the 20th century, almost all the moths in the region were of the form *carbonaria*. In this case, the activity of the species *Homo sapiens* warped the fitness landscape of *Biston betularia* by killing the lichen that it depended on for concealment. *B. betularia* was forced in turn to adapt by evolving a new camouflage.

As a thought experiment to illustrate the point that when new species are added to a cooperative ecosystem the existing species are free to become more specialized, let us say that species A is doing a mediocre job of performing task T. Perhaps it is a generalist and is trying to cover several tasks in addition to T at the expense of performing any of them really well, or perhaps it simply has not had time to evolve a good solution for T. If a new species, B, is created, and one of the individuals of B can perform T better than any individual of species A, the species A individuals who perform T will no longer have a selective advantage over other individuals of A based on their ability to perform that particular skill. As a result, species A will now be free to focus on other skills. Of course, if B can only perform T slightly better than A, we may see the roles quickly reverse. That is, A may produce an individual who through genetic variation is able to assume once again the role of covering T.

This is similar to the notion of *character displacement* that occurs in competitive environments (Brown and Wilson 1956). For example, another 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 competing finch species occupy 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, it 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 are able to adapt to a dynamic environment, we generate a target set from the same three schemata used in the previous section to demonstrate the ability of multiple species to evolve to an appropriate level of generality. We begin this experiment with a single species and add new species on a fixed schedule. Specifically, we add a second species at generation 100 and a

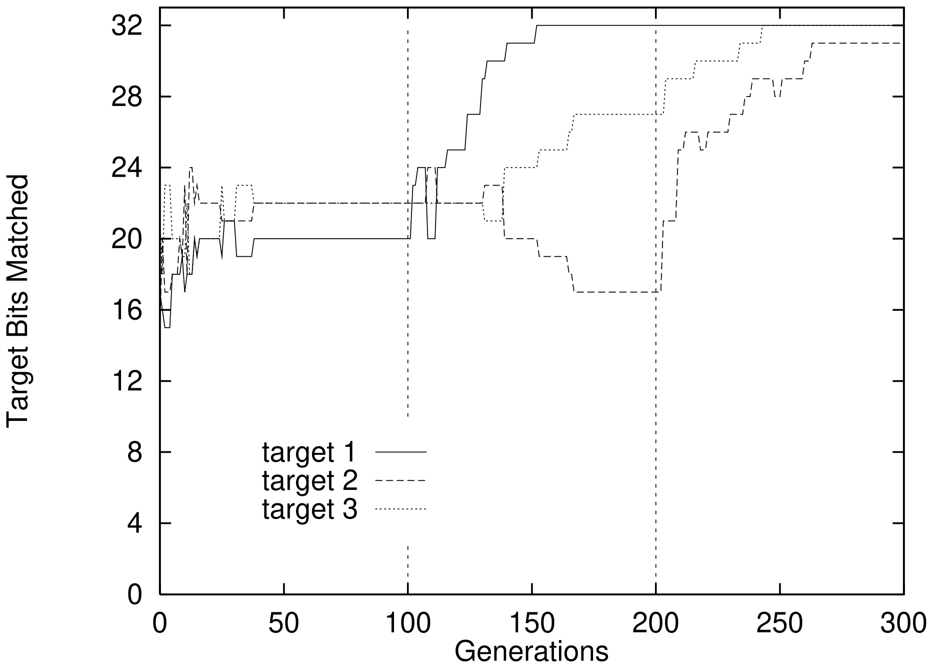


Figure 5.8: Shifting from generalists to specialists as new species are added to the ecosystem on a fixed schedule

third species at generation 200.

The number of target bits from the fixed region of the schemata matched by the best individual from each species is shown in figure 5.8. As in the experiments run in the previous section, the fitness of individuals was based entirely on how well the full 30-element target set of 64-bit strings was covered. Each dashed vertical line marks the creation of a new species. It is clear from the figure that the roles of existing species change as new species are introduced. Furthermore, it is apparent 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. When we compare figure 5.8 with figures 5.3, 5.4, and 5.5 from the previous section, we see that the region of figure 5.8 in which a single species exists is similar to figure 5.3; the region in which two species exists is similar to figure 5.4; and the region in which three species exists is similar to figure 5.5. A final observation is that a period of instability occurs just after each species is introduced. This 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.

5.6 Evolving an Appropriate Number of Species

The fourth and final question we seek to answer in this study of the basic decomposition capability of the model is,

> Will the occasional creation of new species and the elimination of unproductive ones induce the emergence of an appropriate number of species?

It is important not to evolve too many species because each species requires computational resources. This is due to the increasing number of fitness evaluations that need to be performed, to the need for applying operators such as crossover and mutation to more individuals, and to miscellaneous computational overhead such as converting between genotypic and phenotypic representations. 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 sections.

One possible method for evolving an appropriate number of species was illustrated in figure 3.4 on page 34. The basic idea is that we check for evolutionary stagnation by monitoring the change in fitness of the collaborations over time. If we are not improving significantly, the unproductive species are eliminated, and a new species is created.

To determine whether this method is sufficient, we use the same target set as in the previous two sections and begin by evolving a single species. Every ecosystem generation we check for evolutionary stagnation, and if we are not making sufficient improvement, the algorithm for deleting and creating species is applied. In this experiment we define a significant improvement to be an increase in fitness of at least 0.5 over five generations, where the fitness is computed as the match score averaged over the complete set of 64-bit target strings. A unproductive species is defined in this experiment as one who is making a contribution of less than 5.0, where its contribution is defined to be the portion of the collaboration fitness it produces. Therefore, the sum of the contributions from each species is equal to the total fitness of the collaboration. Recall from equation 3.3 on page 40 that a species only contributes to the fitness of a collaboration when it matches a target string better[2] than any other member of the collaboration. We refer to the amount of contribution that a species must make to be considered viable as its *extinction threshold*.

The contributions of each species in the ecosystem over 300 generations are plotted in figure 5.9 on the facing page. The vertical dashed lines represent stagnation events in which unproductive species are eliminated and a new species is created. At generation 139, evolution has stagnated with three species in the ecosystem. The species are contributing 17.13, 16.80, and 15.80 respectively. Of course we know from the experiments in the previous few sections that this is the optimal number of species for this particular problem; however, the macroevolutionary model does not possess this prior knowledge and creates a new species. This species is only able to contribute 1.57 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 1.53, 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; therefore, they are each eliminated in turn when stagnation is detected.

The first observation that can be made from this experiment is that when using our simple method for creating new species and eliminating unproductive ones, an appropriate number of species in the ecosystem emerges. Specifically, the ecosystem stabilizes to a state

[2] Ties are won by the older species.

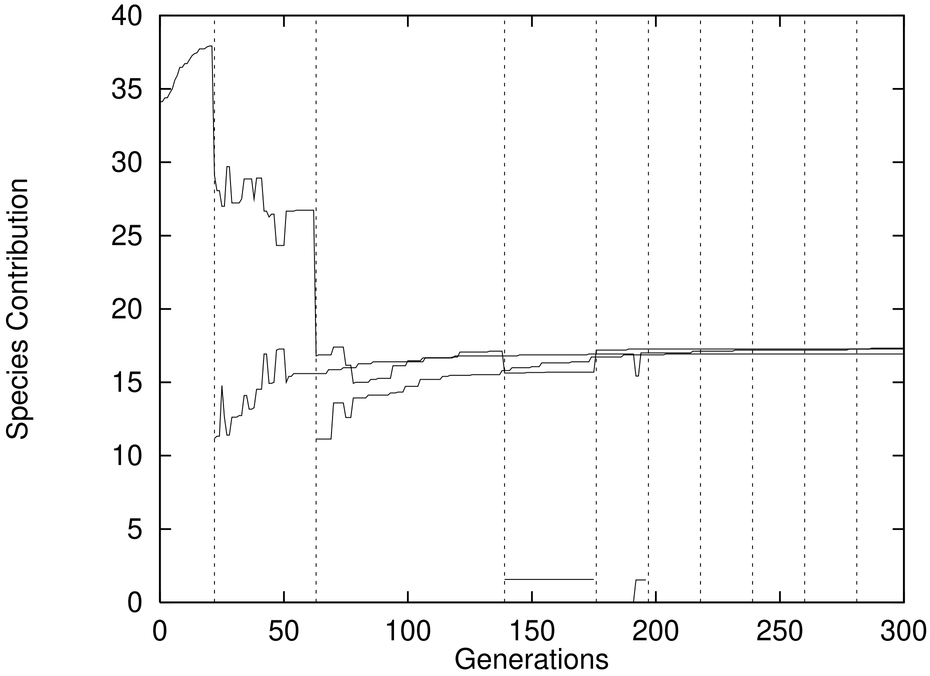


Figure 5.9: Changing contributions as species are dynamically created and eliminated from the ecosystem

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. Although our simple string covering application has a stationary objective function so this insurance is not really necessary, 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. These contributions were the result of repeating 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 these species would have been preserved.

5.7 Summary

In summary, this chapter has demonstrated, within the context of a simple string covering problem, that the model of cooperative coevolution is able to discover important environmental niches, evolve subcomponents appropriate in number and generality to cover those niches, and that the specific roles played by these subcomponents will change as they coadapt to a dynamic fitness landscape. It accomplishes this through a task-independent approach in which the problem decomposition emerges purely as a result of evolutionary pressure to cooperate.

Although the goal of this chapter has been met by providing an initial step in determining whether evolutionary pressure alone is sufficient for producing good decompositions, an application of the model to more complex domains is necessary to determine the robustness of the approach. This and other issues will be explored through a number of case studies in the next chapter.

Chapter 6

CASE STUDIES IN EMERGENT PROBLEM DECOMPOSITION

Now that we have achieved an understanding of the basic decomposition capability of the model, this chapter continues with two relatively complex case studies in which cooperative coevolution is applied to the construction of artificial neural networks and to concept learning. By moving beyond the simple string matching problem of the previous chapter into these more complex domains, we explore the robustness of the model's ability to decompose problems. We are especially interested in determining whether any aspects of the model need to be modified to handle problems that can only be decomposed into subtasks with complex and difficult to understand interdependencies. In the case studies, we also directly compare and contrast the decompositions produced by cooperative coevolution and those produced by some well-known non-evolutionary approaches that are highly task-specific. This comparison provides further insight into the emergence of problem decompositions resulting from the interplay and coadaptation among evolving species sharing a common ecosystem.

Both of the case studies in this chapter follow the same basic format. They begin with a brief description of the domain. This is followed by a section that provides background information on previous approaches to solving problems from the domain with evolutionary computation and a description of our specific approach that applies cooperative coevolution to the task. Next, the non-evolutionary decomposition technique that will be used for comparison is described. This is followed by a description of the specific problem that will be solved. The final section of each case study presents experimental results and observations.

6.1 Artificial Neural Network Case Study

In the first case study, 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. This type of network is typically trained using a gradient-descent technique in which an error signal is propagated backwards through the network (Rumelhart, Hinton, and Williams 1986). The error signal is generally computed as the sum-squared difference between the actual network output and the desired output for each element of a set of training patterns—assuming the desired outputs are known *a priori*. The network is trained to produce the correct output through many iterations of passing training patterns forward through the network, generating and back-propagating the error signal, and appropriately

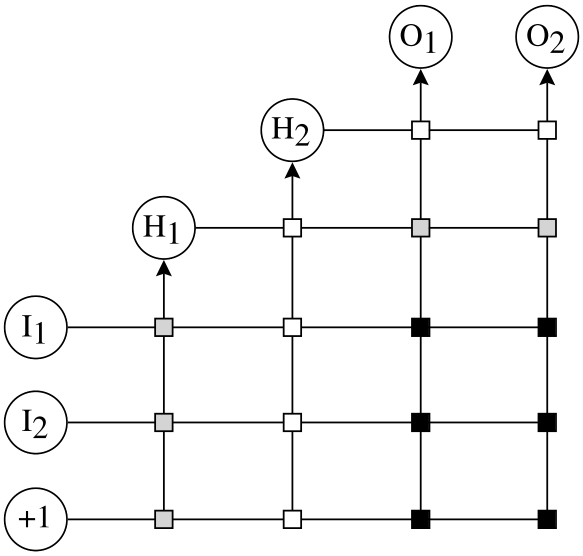


Figure 6.1: Example cascade network

adjusting the connection weights. This form of learning, where the method has access to preclassified training patterns, is called *supervised learning*. If the desired outputs are not known beforehand, as in *reinforcement learning*, where only occasional performance-related feedback is available, an error signal must be computed by some other means. For example, $\mathcal{Q}$-learning—one of the more popular reinforcement learning techniques—estimates the desirability of the network output signal produced by a given input pattern through the use of a predictive function that is learned over time (Watkins 1989; Watkins and Dayan 1992).

6.1.1 Evolving Cascade Networks

A cascade network is a form of multilayered feed-forward artificial neural network in which all input nodes have direct connections to all hidden nodes and to all output nodes. Furthermore, the hidden nodes are ordered and their outputs are cascaded; that is, each hidden node sends its output to all downstream hidden nodes and to all output nodes. An example cascade network having two inputs, a bias signal, and two outputs is shown in figure 6.1. Each box in the figure represents a connection weight. Cascade networks were originally used in conjunction with the cascade-correlation learning architecture (Fahlman and Lebiere 1990). Cascade-correlation, described in detail in section 6.1.2, constructs and trains the network one hidden node at a time using a gradient descent technique. Similar architectures in which a genetic algorithm is used as a replacement for gradient descent have also been explored (Potter 1992; Karunanithi, Das, and Whitley 1992).

Because cascade networks have a well-defined topology, only the weights on the connections and the network size need to be evolved. A direct approach to the evolution of neural network connection weights with a standard evolutionary algorithm is to let each individual represent the connection weight assignments for the entire network. The connection weights may be encoded simply as a vector of floating point numbers if using an evolution strategy, or with a binary encoding in which each real-valued connection weight is mapped to a segment of a string of ones and zeros if using a genetic algorithm. Individuals are typically initialized randomly, and evolved until a network is created with an acceptable level of fitness. The fitness of an individual is determined by computing the sum-squared error as training patterns[1] are fed forward through a network constructed to the individual's specification. Individuals producing networks with lower sum-squared errors are considered more highly fit.

Cascade networks were designed to be constructed one hidden node at a time. A traditional single-population evolutionary algorithm could incrementally construct these networks by initially evolving a population of individuals representing just the direct input-to-output connections. If evolutionary improvement stagnates before a network with sufficiently high fitness has been found, additional random values representing the weighted connections into and out of a new hidden node could be added to the end of each individual. This cycle, of evolving until improvement stagnates and lengthening the individuals, would continue until an acceptable network is found or until a predetermined number of network evaluations have been performed.

For example, if the cascade network shown in figure 6.1 on the facing page was being evolved, the initial population would consist of individuals representing the six connection weights associated with the direct connections, denoted by black boxes, from the input nodes to the output nodes. Note that the bias signal is simply treated as another input. This population would be evolved until improvement approaches an asymptote. Five random values would then be added to the end of each individual to represent the connection weights associated with the first hidden node. These connections are denoted by gray boxes in the figure. Evolution of the population would proceed until stagnation is again detected. At this point, six additional random values would be added to the end of each individual representing the connection weights associated with the second hidden node. These are denoted by white boxes in the figure.

To evolve the cascade network with our computational model of cooperative coevolution, we begin as with the standard evolutionary algorithm by evolving a single species that consists of individuals representing the six connection weights associated with the direct connections from the input nodes to the output nodes. This species would be evolved until improvement stagnates. At this point, a new species would be created whose individuals represent the weights on the three input connections of the first hidden unit. Random values representing the two output connection weights of the hidden unit, denoted by the horizontal pair of gray boxes in the figure, would be appended to the end of each individual belonging to the first species. Now that two species exist, neither the first species nor the second species represents complete networks. To evaluate one of these subnetworks, it will first be combined with an individual from the other species to form a complete network. We use the current best individual from the other species to construct this collaboration. Evolution of these two species proceeds in parallel until improvement again approaches an asymptote. At this point a third species is created whose individuals represent the four input connection weights associated with the second hidden unit. As before, the individuals in the first species will be lengthened by two in order to represent the output connection weights of the second hidden unit. This cycle continues until a network is created that produces a sufficiently low sum-squared error. Using this scheme, a cascade network with

[1] Supervised learning is assumed.

k hidden nodes would be constructed from $k+1$ species.

The specific evolutionary algorithm used in this case study is a (μ, λ) evolution strategy[2] as described in figure 2.3 on page 13. That is, we have implemented a *coevolution strategy*. In our experiments, $\mu = 10$ and $\lambda = 100$. 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 and initialized as described by equations 2.2, 2.3, and 2.4. The constants C and R are set to one and twenty respectively. Mutation is the only evolutionary operator used. Connection weights are limited to the range $(-10.0, 10.0)$ and are randomly initialized.

Along with using an evolution strategy instead of a genetic algorithm, there is one additional difference between the cooperative coevolutionary system used in this case study and the one used in studying the basic decomposition capability of the model. Here we use a slightly different approach when creating and eliminating species. A species is created, as before, when evolutionary improvement stagnates; however, once it is created, we allow just it and the species representing the weights on the connections to the output units to be evolved until progress again approaches an asymptote. At this point, we will either eliminate the new species if it is not making a significant contribution and create another one to replace it, or we will continue with the evolution of *all* of the species in the ecosystem. This small modification focuses our computational resources on enabling new species to find a niche in which they can contribute more quickly—a change we found necessary due to the greater complexity of the neural network search space.

6.1.2 The Cascade-Correlation Approach to Decomposition

In the context of a cascade network, problem decomposition consists of determining how many hidden nodes are required and what purpose each hidden node will serve. We will be comparing and contrasting the decompositions produced by cooperative coevolution to those produced by cascade-correlation—a statistical technique designed specifically for cascade networks by 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 choose a reasonable topology, that is, the number of hidden units, layers, and connectivity. The roles of the hidden units were then allowed to emerge through the application of the back-propagation algorithm. One source of inefficiency Fahlman and Lebiere noticed in this process was what they called the "moving target problem". They made the following observation:

> Instead of a situation in which each unit moves quickly and directly to assume some useful role, we see a complex dance among all the units that takes a long time to settle down.

The cascade-correlation learning architecture was designed to eliminate the "complex dance" observed by Fahlman and Lebiere by constructing the network one hidden unit at a time and freezing the roles of the hidden units once established. The algorithm begins with a network composed of only input and output units as described in the previous section. The connection weights of this smallest-possible network are trained with the *quickprop*

[2]We have also applied a genetic algorithm to this task (Potter and De Jong 1995), but achieved better results with the evolution strategy.

algorithm—a second-order technique related to Newton's method (Fahlman 1988)—using the sum-squared error as preclassified training patterns are fed forward through the network. When improvement approaches an asymptote, a single hidden unit is added by way of a two-phase process. In the first phase, the hidden unit[3] is partially "wired" into the network. Specifically, the hidden unit receives input signals but does not contribute to the network output. The weights on its input connections are trained with quickprop using 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. In other words, the hidden unit is trained to respond either positively or negatively to the largest portion of remaining error signal. In practice, 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 training approaches an asymptote, the input weights are frozen and the hidden unit is fully connected. We then enter the second phase in which the output connection weights are trained as before. This cycle of adding a new hidden unit, training and freezing its input connection weights, and training the entire set of output connection weights will continue until a sufficiently low sum-squared error is produced.

6.1.3 Two-Spirals Problem

We will construct cascade networks to solve the *two-spirals problem*. The two-spirals problem, originally proposed by Alexis Wieland (posted to the *connectionists* mailing list on the internet), 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 clearly 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 back propagation; therefore, the problem has been used in a number of previous studies to test new network learning methods (Lang and Witbrock 1988; Fahlman and Lebiere 1990; Whitley and Karunanithi 1991; Suewatanakul and Himmelblau 1992; Potter 1992; Karunanithi, Das, and Whitley 1992).

To learn to solve this task, we are given a training set consisting of 194 preclassified coordinates as shown in figure 6.2 on the next page. Half of the coordinates are located in a spiral-shaped region designated as the *black spiral* and the other half of the coordinates are located in an interlocking spiral-shaped region designated as the *white spiral*. The 97 black spiral coordinates are generated using the following equations:

$$r = \frac{6.5(104 - i)}{104} \tag{6.1}$$

$$\theta = i\frac{\pi}{16} \tag{6.2}$$

$$x = r\sin\theta \tag{6.3}$$

$$y = r\cos\theta \tag{6.4}$$

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

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

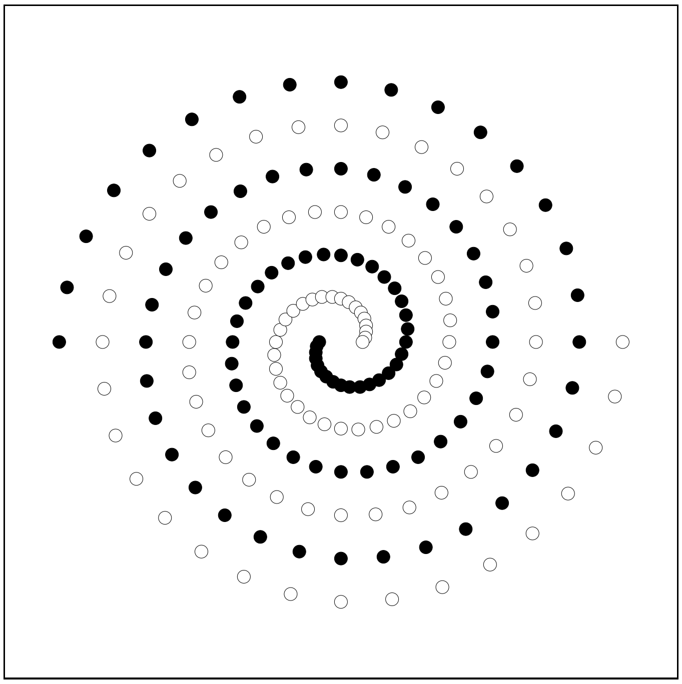


Figure 6.2: Training set for the two-spirals problem

When performing a correct classification, the neural network takes two inputs corresponding to an (x, y) coordinate, and produces a $+0.5$ if the point falls within the black spiral region and a -0.5 if the point falls within the white spiral region.

6.1.4 Experimental Results

Ten runs were performed using the cascade-correlation algorithm and an additional ten runs were performed using cooperative coevolution. The algorithms were terminated when all 194 training patterns were classified correctly as belonging to the black spiral or white spiral. We first look at the number of hidden units produced by each method and then use a couple of different visualization techniques to gain an understanding of what roles the hidden units are performing.

Over ten runs, the cascade-correlation algorithm generated networks capable of correctly classifying all the two-spirals training patterns. As shown in table 6.1 on the facing page, the networks required an average of 16.8 hidden units. The table includes 95-percent confidence intervals on the mean computed from the t-statistic. These results are consistent with those reported by Fahlman and Lebiere (1990). In contrast, cooperative coevolution was only able to generate a network capable of correctly classifying all the training patterns in seven out of ten runs. However, in the seven successful runs, the networks produced by cooperative coevolution required an average of only 13.7 hidden units to perform the task. Although this represents a statistically significant difference in the number of hidden units required to solve the problem, the parameters of the cascade-correlation algorithm could probably be tuned to favor networks with fewer hidden units at the expense of an increased number

Table 6.1: Required number of hidden units

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

of training epochs. The p-value produced from a two-sided 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 6.2 on page 98 was generated by starting with the final networks produced by the first runs of the two methods and eliminating one hidden node at a time while measuring the number of training-set misclassifications and the sum-squared error. The first run was chosen for this comparison arbitrarily; however, it appears to provide a reasonably fair comparison. The data is presented in the reverse order from how it was gathered—beginning with a network containing no hidden units and adding one unit at a time. 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 and contain fewer elements. As we previously mentioned, this difference could probably be eliminated by tuning the parameters of the algorithms.

We continue with our characterization of the roles played by the hidden units produced by the two methods by studying a series of *field-response diagrams* generated from the same networks summarized in table 6.2. The field-response diagrams shown in figures 6.3 and 6.4 were produced from the cascade-correlation network, and those shown in figures 6.5 and 6.6 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 6.3 on the following page the bottom-left pair of field response diagrams is generated from a cascade-correlation network in which all but the first six hidden units have been eliminated. The left diagram of that particular pair shows the output from the sixth hidden unit and the right diagram of the pair shows the corresponding output from 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. Some of this symmetry

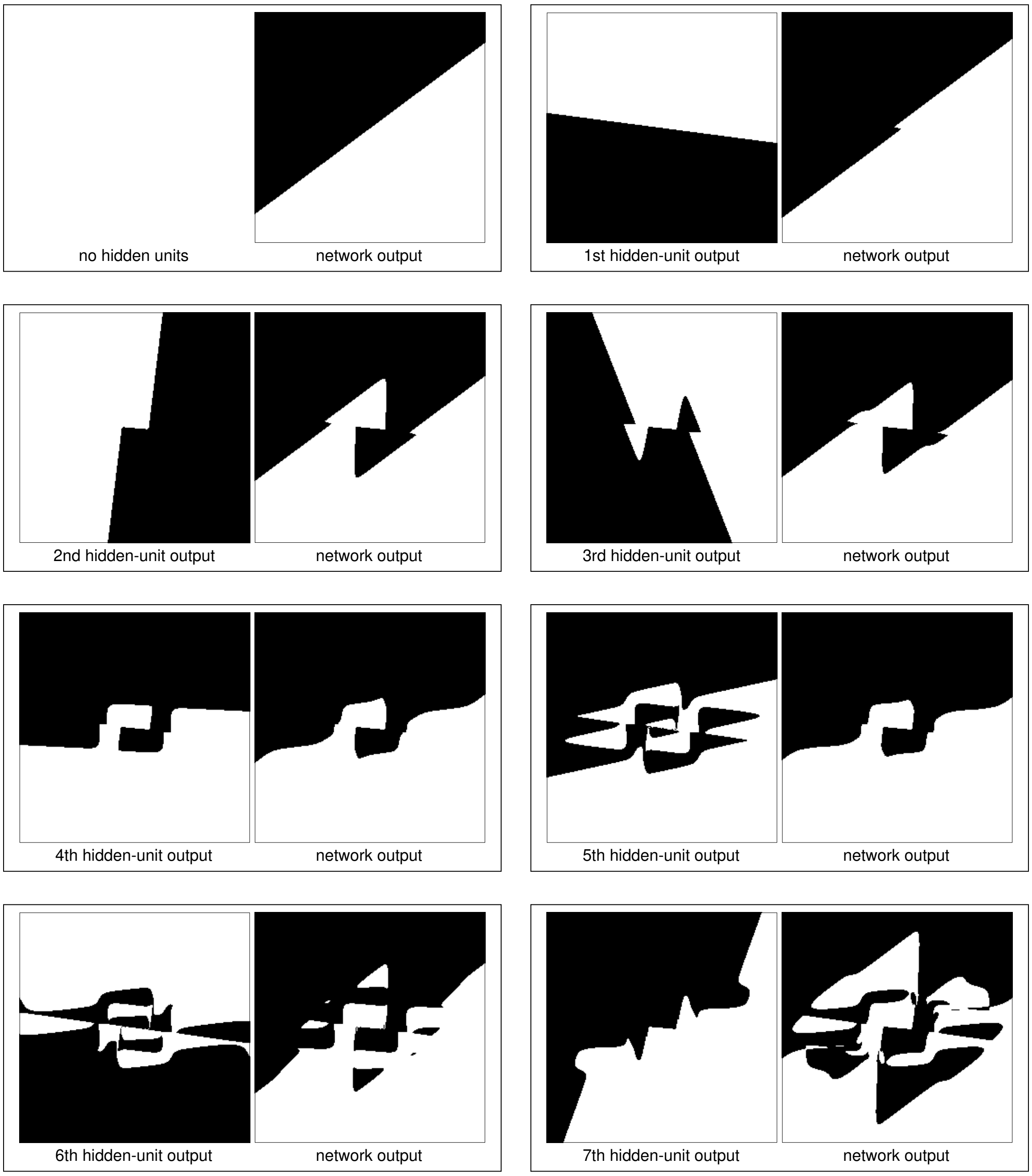


Figure 6.3: Effect of adding hidden units on field response of network generated with cascade-correlation algorithm

Figure 6.4: Effect of adding hidden units on field response of network generated with cascade-correlation algorithm (continued)

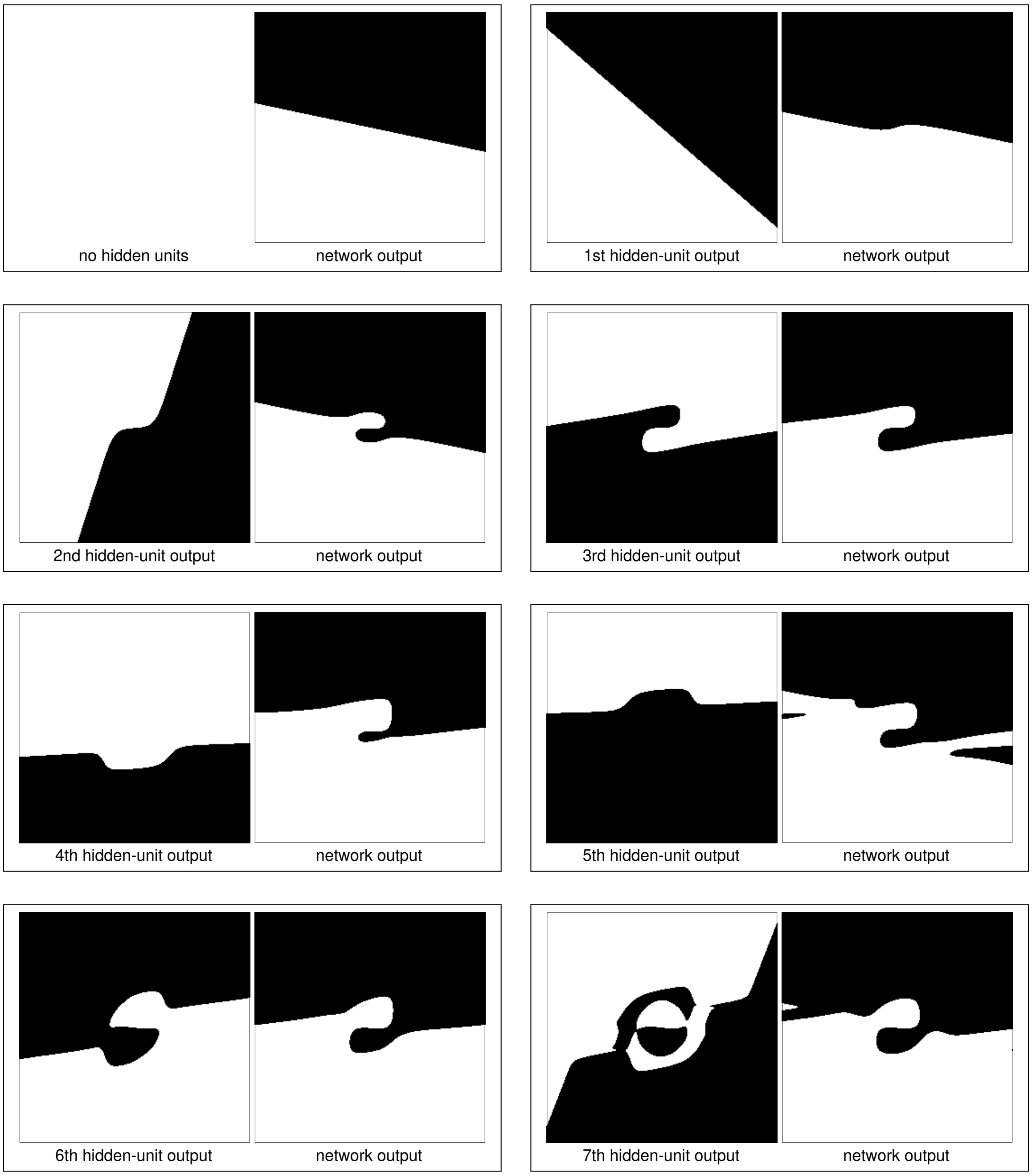


Figure 6.5: Effect of adding hidden units on field response of network generated with cooperative coevolution

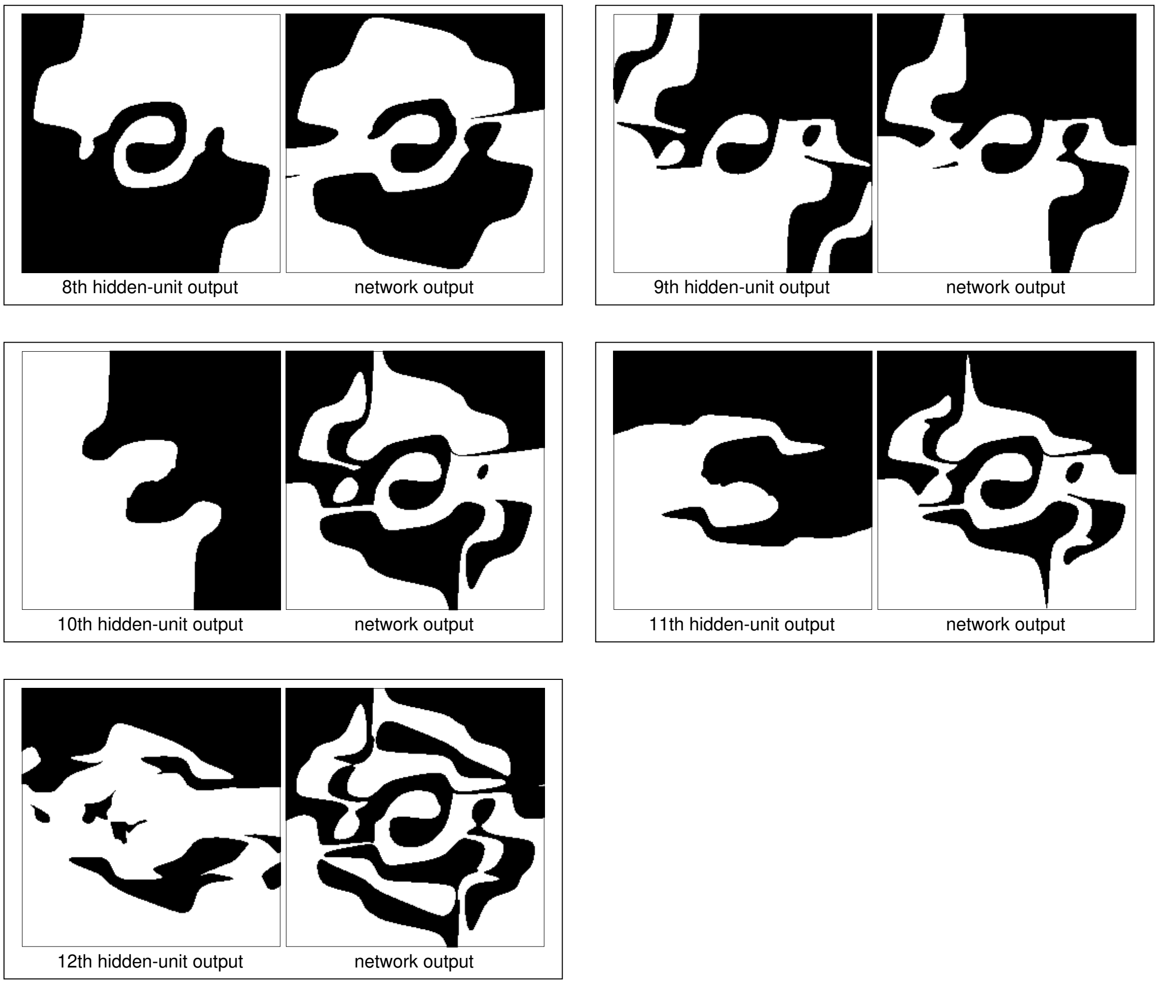


Figure 6.6: Effect of adding hidden units on field response of network generated with cooperative coevolution (continued)

Table 6.2: 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	

is due to the white spiral coordinates being the negation of the black spiral coordinates. 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 fact that the training set elements are more concentrated in the center of the field, as one can see from figure 6.2 on page 92. A third similarity is that as hidden units are added to the network, their response patterns tend to become increasingly complex, 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 6.1 on page 88.

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 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 node and the network error signal to train the weights on the connections leading into the node. This enables the hidden node to respond precisely to the (possibly) few 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 cooperative coevolution does not use task-specific statistical information as a focusing tool, it tends to paint with broader brush strokes.

The obvious disadvantage of cascade-correlation is that it assumes not only that the task is to build a cascade network, but also that a set of preclassified training patterns is available. Cooperative coevolution does not make these assumptions; therefore, it has a much wider range of applicability. For example, it could be effectively applied to problems in reinforcement learning—an area in which genetic algorithms have proven to be superior to current non-evolutionary techniques for training neural networks (Moriarty and Miikkulainen 1996). Its disadvantages with respect to supervised learning, however, are that it is much slower and sometimes is not able to drive the misclassification rate completely down to zero.

In summary, this case study demonstrates the emergence of good decompositions when using cooperative coevolution in the complex domain of artificial neural network construction. We again emphasize that the advantage of cooperative coevolution over other methods in this domain is its generality. The case study has a side benefit of demonstrating that cooperative coevolution is as applicable to evolution strategies as it is to genetic algorithms. Although we have only used cooperative coevolution in conjunction with genetic algorithms and evolution strategies in our experiments, we see no reason why this meta strategy would not be applicable to other evolutionary algorithms such as evolutionary and genetic programming.

The case study also uncovers a limitation of the model. In all three unsuccessful runs, failure occurred for the same reason. After all but a few of the 194 training patterns were correctly classified, the new species being created were unable to find a niche in which they could contribute. Until a niche is found, no member of the population is distinguished from any other; that is, each will have a fitness of zero. From the basic principles of Darwinian evolution, a population only adapts when variation among individuals produces a selective advantage. Further investigation of the three 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.

6.2 Concept Learning Case Study

In this case study, our task will be to construct a general description of a concept from a set of preclassified positive and negative examples. As in the previous case study, this is an example of a supervised learning task. Once we have learned a concept description, we should be able to determine correctly whether previously unclassified examples are instances of the concept. We say that the task is to construct a *general description* because it should cover unclassified examples that are different from any of the preclassified examples used for learning.

Concept learning is a task that has been extensively studied by researchers in the field of machine learning. Much of this work has been in the area of inductive learning from examples using symbolic representation languages such as predicate calculus (Michalski 1983) and decision trees (Quinlan 1986). Other approaches are also possible. For example, in this case study we will be experimenting with a biologically inspired representation in which concept descriptions are evolved using a model of the immune system.

6.2.1 Evolving an Immune System for Concept Learning

Most of the previous work in which evolutionary computation has been applied to concept learning has taken the approach of evolving binary-string genotypic representations with a genetic algorithm and mapping them into some form of symbolic phenotypic representation for evaluation, such as propositional logic. For some examples of this approach, see (Janikow 1991; Janikow 1993; De Jong, Spears, and Gordon 1993; Giordana, Saitta, and Zini 1994). Here we take a radically different approach in which we use a simple model of one of the recognition processes occurring within the vertebrate immune system to distinguish between concepts. The motivation behind this approach is that the immune system has a highly developed ability to discriminate between self and non-self, that is, to distinguish between the vast array of molecules that are an integral part of our bodies and foreign molecules. We have already seen two examples (evolution and neural networks) of successfully applying computational models of biological systems to the solution of technical problems. We believe that the immune system, especially some of its adaptive components, represents a third example of a biological process that can be modeled and applied to a variety of problems in which the ability to discriminate is required.

We begin with a brief description of the immune system. This is followed by a more specific description of how we model one of its subsystems using cooperative coevolution and how we apply the model to the problem of concept learning. Previous work on building computational models of the immune system has been described earlier in this dissertation.

An Overview of the Immune System

The purpose of the immune system is to protect our bodies from infection. The system works by recognizing the molecular signature of microbes or viruses that attack our bodies, and once identified, eliminating the foreign molecules in a variety of ways. The immune system consists of two interrelated components: an innate defense component and an adaptive component. Here we will focus on the adaptive component, which is responsible for acquired immunity.

We call the molecules capable of stimulating an acquired immune response *antigens*. When the system is working properly, only foreign antigens will produce an immune response. There are a number of ways antigens are recognized, depending on whether the foreign molecule is inside or outside one of our body's own cells. It is the job of *antibodies*—protein molecules displayed on the surface of a type of white blood cell produced in the bone marrow called a B-lymphocyte—to recognize antigens that are located in our body fluid outside the cell boundary. Recognition by a B-lymphocyte occurs when one of its antibodies comes into contact with an antigen of complementary shape. Although all the antibodies on an individual B-lymphocyte have the same shape, we have about 10 trillion of these B-lymphocytes circulating throughout our body, and they collectively have the potential of representing about 100 million distinct antibody molecules at any one time. If the B-lymphocyte recognizes an antigen, it develops into a *plasma cell* and begins excreting large quantities of the antibody. The antibody, now circulating freely in the serum, coats foreign molecules of like type and flags them for destruction. The flagged molecules may be consumed, for example, by scavenger cells such as *macrophages*. In addition, the activated B-lymphocytes enter a phase of hypermutation. The effect is to create offspring—called *clone cells* because they come from one parent—that produce antibody with an even greater affinity to bind to the specific type of foreign molecule under attack.

The antibody molecules are composed of two pairs of protein chains: the so-called heavy chains and light chains. The heavy chains are constructed from four families of genes called variable (V), diversity (D), joining (J), and constant (C). While each of these gene families has a number of members, only one gene from each family—along with additional random DNA segments—is used in constructing the protein. The chosen gene from each family is not determined until the antibody is being formed, thereby enabling a few hundred genes to create thousands of different heavy chain types through combinatorics. Similarly, the light chains are constructed from the V, J, and C families. Because it is the specific combination of light and heavy chains that determines what form of antigen the antibody will recognize, the potential coverage is around 100 million distinct foreign molecules.

Another type of white blood cell—called a T-lymphocyte because it is produced in the thymus gland—is able to recognize foreign molecules, such as viruses, that take up residence within the body of our own cells. This is a more complex recognition process in which protein fragments (peptides) of the invader are carried to the surface of the cell in which they are hiding by the molecule *major histocompatibility complex* (MHC). The T-lymphocytes display receptors on their surface that are sensitive to a specific peptide-MHC complex, and they are constructed and function similarly to the receptors on the surface of the B-cells. Therefore, once the foreign peptides are transported outside the cell membrane by MHC, the T-lymphocytes are able to recognize them and launch an attack. The specific nature of the attack depends on whether the peptide-MHC complex is recognized by a *helper* T-cell or a *killer* T-cell. The killer T-cells respond to so-called class I MHC by attaching themselves to the infected cell and attacking it directly. The helper T-cells respond to so-called class II MHC, which is only produced by macrophage cells, by sending out a chemical messenger called *cytokines* that stimulates the macrophage to destroy the parasite hiding within it. Helper T-cells also play a role in the stimulation of B-lymphocytes to begin secreting antibodies.

One should realize that the immune system is quite complex and is the focus of much current research. We have only provided a very brief overview of some of its processes here. Although this description should be sufficient for an understanding of this case study, for more details concerning the workings of the immune system, see, for example, (Roitt 1994).

A Cooperative Coevolutionary Model of the Immune System

As in previous evolutionary computation models of the vertebrate immune system (cf. Forrest and Perelson 1990), our model is limited to the interaction between B-lymphocytes and antigens. It evolves these entities with a coevolutionary genetic algorithm similar to the implementation described in chapter 3, and uses binary strings to represent their genetic codes[4]. Some of the other details of the implementation used in this case study include: a population size of 100, random initialization, uniform crossover at a rate of 0.6, and a

[4] In this simple model, little distinction is made between antibodies and antigens and the cells on which they are displayed. We will use the terms B-lymphocyte or B-cell when we are referring to the combination of a receptor (antibody) and an activation threshold.

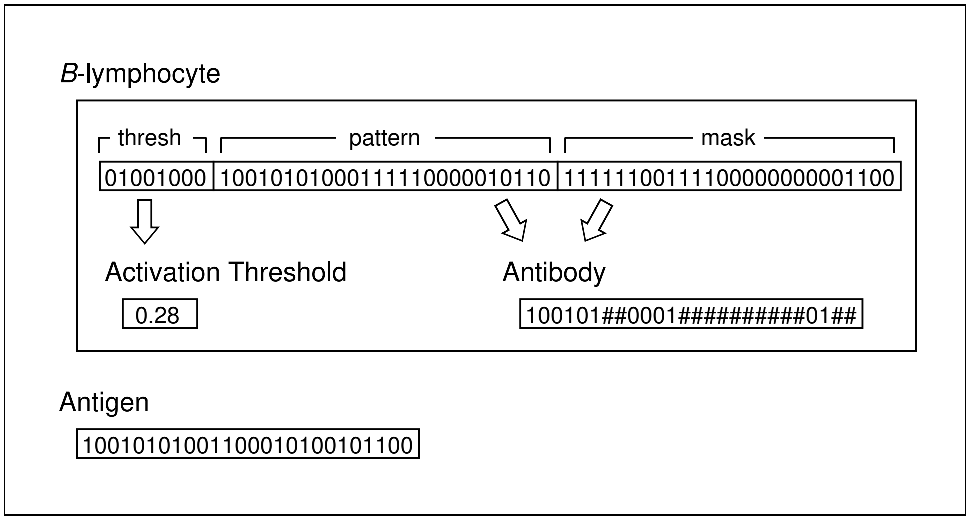


Figure 6.7: B-lymphocyte and antigen representations

bit-flipping mutation rate set to twice the reciprocal of the chromosome length.

In biological systems, antibodies and antigens are folded into complex three-dimensional shapes. The closer the complementary match between their shapes, the stronger the binding forces will be between them. To represent these molecules using a genetic algorithm, one possibility is to make no distinction between their genotype and their phenotype, that is, simply to represent both antigens and antibodies as binary strings. Given this type of representation, the binding force between a particular antibody and antigen can be computed simply as a function of the similarity between their sequences of ones and zeros. However, we use a slightly more complex schema representation for antibody phenotypes to enable some regions of the receptor protein chains to be ignored in determining their final geometric shape. This gives us the ability to model a range of antibodies from specialists, which can only bind to a specific antigen, to more general antibodies, which can bind to whole families of antigens that share common characteristics. In addition to antibody genes, each B-lymphocyte has a "threshold gene" that represents the binding strength required to initiate an immune response. Our representation of both B-lymphocytes and antigens is shown in figure 6.7. Note that we produce an antibody—represented as a trinary schema—from a binary pattern and mask gene. A mask bit of one generates a schema value equal to the corresponding pattern bit, while a mask bit of zero produces a "don't care" schema value. The length of the pattern and mask genes depends on the complexity of the antigens the antibody must recognize. The real-valued activation threshold of the B-lymphocyte, in the range $[0, 1]$, is produced from an 8-bit threshold gene. There is no distinction made between the genotype and phenotype of an antigen.

In our model of coevolution, each species represents a population of B-lymphocytes in one of three emergent phases of development. During the first phase—which begins immediately after the species is created and continues until some of its B-cells are activated by antigens—no cell has a selective advantage over any other so they are all reproducing at a uniformly slow rate. Once some of the B-cells are activated by antigens, these cells begin rapidly reproducing—marking the beginning of the second phase. This is also a time when large changes in fitness occur as the crossover operator splices pieces of various successful B-cells together. Eventually, the population will converge to slight variations of the most highly fit B-cell and enter a third phase. Mutation is the dominant genetic operator during this third phase, and it will produce relatively slight changes in cell fitness.

The B-cell development phases of our model differ somewhat from those in nature. Recall from the immune system overview that when an actual B-cell is activated, it enters a state of hypermutation called clonal selection. However, in both nature and our model the activation of a B-cell marks the beginning of a period of rapid change.

Evolution begins with a single species. New species are created and unproductive species eliminated as illustrated in figure 3.4 on page 34 when evolutionary improvement stagnates, as determined by equation 3.1 on page 34. Evolutionary stagnation generally occurs after the most recently created species has entered its third phase. In the context of this model, problem decomposition consists of determining how many B-cells are required to cover a set of antigens, and which antigens will be recognized by which B-cells.

The fitness of a B-lymphocyte is computed by adding it to a "serum" consisting of the current best B-cells from each of the other species in the ecosystem. Each member of a set of antigens (both foreign and self) is then presented to the serum. A particular B-cell is considered to have recognized an antigen if the binding strength between its antibody and the antigen exceeds its activation threshold and the antigen binds to the antibody more strongly than to any other antibody in the serum. The fitness of the B-cell is defined to be the number of foreign antigens recognized by all the antibodies in the serum, minus the number of false-positives, that is, self antigens flagged as foreign. Therefore, as in our other instantiations of cooperative coevolution, each B-cell is rewarded based on how well it collaborates with B-cells from each of the other species to cover the collection of antigens.

A linear matching function that returns the percentage of matching bits in the antibody and antigen vectors is used to compute the binding strength. The locations at which the antibody contains a "don't care" are ignored. We also experimented with a variety of matching functions that are biased toward longer sequences of matching bits; however, these more complex matching functions produced no significant performance improvement.

This model is applied to concept learning from preclassified positive and negative examples by having the set of positive examples represent foreign antigens and the set of negative examples represent self antigens. Once the fitness of the immune system increases to a point where all of the foreign antigens and none of the self antigens are recognized, the best antibodies from each species collectively represent a description of the concept[5]. This model can easily be generalized to discriminate between more than two classes by evolving a separate family of antibodies for each concept. Each family would then recognize examples of one concept as foreign and all the other examples as self. In this way, k classes could be covered by $k-1$ families of antibodies. This could be simply implemented by adding a "class gene" to the B-lymphocytes.

What we have described in this section is admittedly an extremely loose model of an actual vertebrate immune system. We emphasize that the focus of this chapter is on emergent problem decomposition—not biology. It is our belief, however, that there is a potential for

[5]Given noisy examples, the immune system would be evolved until *most* of the foreign antigens and *few* of the self antigens are recognized.

$ConceptDescript$ = nil
WHILE $ConceptDescript$ does not cover all positive examples **BEGIN**
Randomly select an uncovered positive example P_k
Compute a bounded star that covers P_k without covering any negative examples
Select a single conjunctive description C from the bounded star according to user-supplied preference criteria
$ConceptDescript \leftarrow ConceptDescript \vee C$
END
RETURN $ConceptDescript$

Figure 6.8: AQ algorithm

building more biologically faithful coevolutionary models of the immune system that may lead, not only to better machine learning systems, but also to greater insight into the workings of our bodies. This issue will be further addressed in the final chapter.

6.2.2 The AQ Approach to Decomposition

We will be comparing the decompositions produced by our cooperative coevolutionary immune system model with those produced by AQ15, a symbolic inductive learning system developed by Ryszard Michalski et al. (1986). This system is one of the latest in a series of AQ systems that constructs conjunctive descriptions using an enhanced propositional calculus representation language. A complete AQ concept description consists of a disjunction of conjunctive descriptions. In the context of AQ, problem decomposition consists of determining how many conjunctive descriptions are required to cover a set of training examples and how the example set will be partitioned by the descriptions; that is, which positive examples will be covered by each of the conjunctions.

Problem decomposition is accomplished in AQ by repeatedly applying a task-specific technique called the *star methodology*. A star is the set of the most general conjunctive descriptions that cover one of the positive examples without covering any of the negative examples. Since in complex domains the size of the stars can become unmanageable, they are bounded by applying user-supplied preference criteria to eliminate some of the descriptions. These *bounded stars* are repeatedly constructed until all positive examples are covered. A concept description in disjunctive normal form is progressively built, one disjunct at a time, by combining a single conjunctive description from each bounded star. The conjunctive descriptions are chosen by applying user-supplied preference criteria—as was done in bounding the stars. The complete decomposition algorithm is shown in figure 6.8. The '$\leftarrow$' operator in the figure represents substitution. The preference criteria used in this case study for selecting the conjunctive descriptions are to maximize the number of newly covered positive examples and to minimize the number of conjuncts.

Table 6.3: Issues voted on by 1984 U.S. House of Representatives

Index	*Issue*
1	handicapped infants
2	water project cost sharing
3	adoption of the budget resolution
4	physician fee freeze
5	el salvador aid
6	religious groups in schools
7	anti satellite test ban
8	aid to nicaraguan contras
9	mx missile
10	immigration
11	synfuels corporation cutback
12	education spending
13	superfund right to sue
14	crime
15	duty free exports
16	export administration act south africa

6.2.3 Congressional Voting Records Data Set

In this case study we will evolve a political party classification system for members of the U.S. House of Representatives given their voting records; that is, we will learn to discriminate between the concepts *republican* and *democrat*. As in the neural network case study, this is a supervised learning task in which we are given a number of preclassified training examples. The data set from which the training examples are drawn consists of 435 voting records (267 democrat and 168 republican). Each record gives the vote cast by an individual on the 16 issues shown in table 6.3. Although the actual voting records are somewhat more complex, each vote in the compiled data set has been simplified to either a yea, nay, or abstain. For use by our cooperative coevolutionary immune system model, the symbolic voting records were converted into 32-bit binary strings (antigens) using the mapping shown in table 6.4 on the next page. Depending on one's political orientation, the foreign antigens to be targeted by the immune system could represent either examples of republicans or democrats. The symbolic data set was originally used in a machine learning study by Schlimmer (1987) and was compiled from actual voting records from the 98th Congress (1984).

6.2.4 Experimental Results

We first look at the quality of solutions produced by the coevolutionary immune system model and the AQ algorithm in terms of how well they are able to discriminate between republicans and democrats. As in the previous case study, a number of different visualization

Table 6.4: Mapping between voting records and binary strings

Vote	*Binary pattern*
abstain	00
yea	01
nay	10

techniques will then be used to compare and contrast the decompositions produced.

Alternative methods for supervised concept learning are generally compared using a metric called *predictive accuracy*. The converse of the predictive accuracy metric is an estimate of the *true error rate*, which is defined to be the rate of errors the classifier would produce if it were tested on a true distribution of examples from the "real world". This can be accurately estimated with a set of several thousand unbiased examples; however, in the case of the voting records data set we are limited to a set of only 435 examples. The tenfold cross-validation method is the recommended procedure for computing predictive accuracy when more than 100 examples are available (Weiss and Kulikowski 1991). One performs tenfold cross validation by randomly dividing the complete set of positive and negative examples into ten partitions of approximately equal size. Ten runs are then performed, each using a different set of nine partitions as the *training set* and the remaining partition as the *testing set*. During each run, the concept learner will use the training set to construct a concept description. Once the run is complete, the concept description is applied to the testing set and the correct-classification rate is computed. The predictive accuracy is computed by averaging the correct-classification rate produced from the ten runs.

Predictive accuracy curves for two variations of the immune system model evolved to recognize democrats and ignore republicans are shown in figure 6.9 on the facing page. We emphasize that the predictive accuracy measure was not used in any way by the co-evolutionary system to influence the development of the B-cells; that is, only the training examples were used in evaluating fitness. We compute the predictive accuracy using the testing examples at the end of each generation purely for use in a post-mortem analysis of the effectiveness of the method. In the first variation, which is labeled *unbiased*, the B-lymphocyte mask genes were initialized completely randomly; while in the second variation, labeled *biased*, approximately 90 percent of the alleles of each mask gene were initialized to zero. Since a mask allele of zero generates a "don't care" in the antibody phenotype, the biased masks produced populations of more general antibodies. The runs were terminated after 100 generations—enough time for the development of B-cells capable of correctly classifying about 97 percent of the training examples. From the graph it is apparent that by biasing the system to evolve more general receptors, fewer generations are required for the populations of B-cells to achieve a high level of competence in distinguishing self from non-self. Specifically, the biased version achieved a predictive accuracy greater than 0.94 in only two generations, while the unbiased version required 53 generations to reach a level greater than 0.94. However, when we compare the biased and unbiased versions at the end of the runs, we see almost no difference in predictive accuracy.

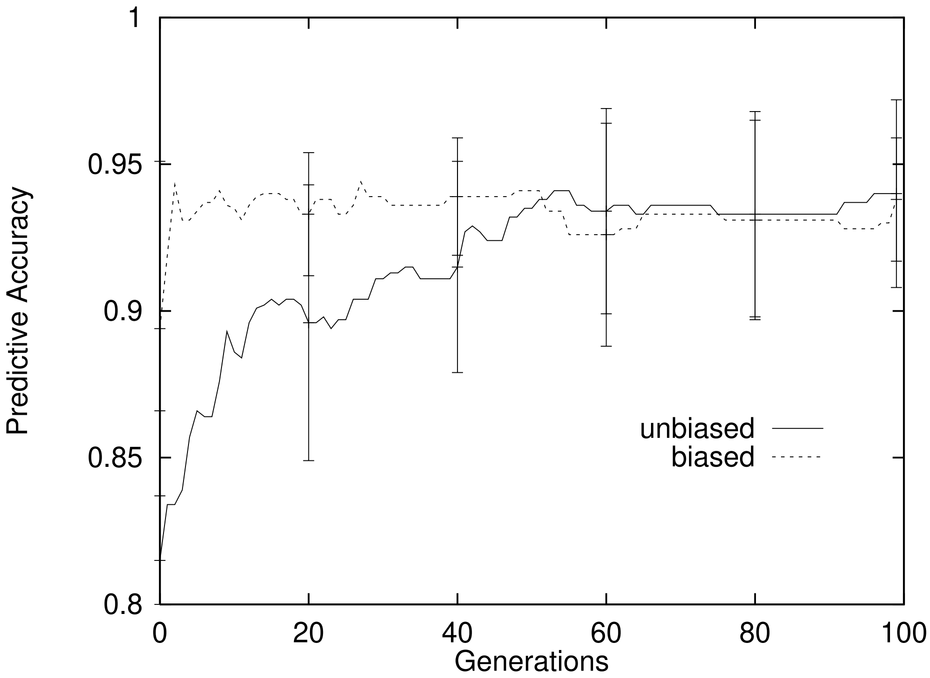


Figure 6.9: Effect of initial bias on predictive accuracy of immune system model

The predictive accuracy of AQ15 on the voting records data set was also computed. The AQ system was terminated when all the instances of republican and democrat voting records in the training set were classified correctly. Rather than simply learning a concept description for one of the classes, as is done in the immune system model, AQ15 learns a separate concept description for each class. For example, it will first use the algorithm shown in figure 6.8 on page 104 to learn a concept description for republicans using the republican instances in the training set as positive examples and the democrat instances as negative examples. It then will learn a concept description for democrats using the opposite orientation. Later, when classifying the examples in the testing set, a conflict resolution procedure will be used if the republican and democrat concept descriptions both match the same example. We can also use this technique in conjunction with the immune system model by evolving two distinct classes of B-cells. One class will recognize democrats (non-self) and ignore republicans (self), while the other family will ignore democrats (self) and recognize republicans (non-self). Each time cooperative coevolution creates a new species of B-cells, it will have the opposite political party orientation of the previously created species.

The final predictive accuracy results from AQ15, the biased and unbiased single-class immune system models, and the biased and unbiased two-class immune system models are summarized in table 6.5 on the next page. The table includes 95-percent confidence intervals on the predictive accuracy measure computed from the t-statistic. A one-way analysis of variance (ANOVA) was also run on the predictive accuracy results from the five methods and no statistically significant difference between the means was found. The p-value from the ANOVA was 0.271.

Table 6.5: Final predictive accuracy comparison of learning methods

Learning method	*Predictive accuracy*
One cell class	
unbiased	0.940 ± 0.032
biased	0.938 ± 0.021
Two cell classes	
unbiased	0.935 ± 0.027
biased	0.964 ± 0.018
AQ	0.956 ± 0.023

Table 6.6: Required number of cover elements

Method	*Elements*		
	Mean	*Min*	*Max*
Coevolution	5.10 ± 0.79	7	4
AQ	8.30 ± 0.68	9	6

The biased single-class variation of the immune system model will be used to compare and contrast the decompositions produced by coevolution with those produced by the AQ system. We choose the biased (for generality) variation of the immune system model because AQ is also biased to produce the most general descriptions possible. The single-class variation of the immune system model is chosen for two reasons. First, for simplicity, we will only be analyzing concept descriptions of democrats; therefore, only the results from the AQ iterations in which democrat instances are taken to be positive examples and republican instances are taken to be negative examples are relevant. Given that we will be analyzing a single-class AQ decomposition, a comparison with a single-class immune system decomposition is the most meaningful. Second, the single-class variation of the immune system model is more biologically faithful.

We first contrast the number of components in the decompositions produced by the two methods, specifically, the number of B-cells versus the number of conjunctive descriptions required to cover the voting record training examples. Over ten runs, the immune system consistently produced smaller covers than AQ. As shown in table 6.6, the immune system model produced final-generation covers consisting of an average of 5.10 B-cells. The table includes 95-percent confidence intervals on the mean computed from the t-statistic. In contrast, the AQ system generated covers consisting of an average of 8.30 conjunctive descriptions. This represents a statistically significant difference between the methods. A two-sided t-test of the means produced a p-value of 0.0000.

Table 6.7: Interpretation of antibody schema

Schema	*Interpretation*
00	abstain or half credit for yea or nay
01	yea or half credit for abstain
10	nay or half credit for abstain
11	half credit for yea or nay
0#	abstain or yea
1#	nay
#0	abstain or nay
#1	yea
##	ignore

To characterize the roles played by the components of the decompositions, the solutions produced by both methods were converted into similar rule-based representations. The schema-based antibodies evolved by the immune system model were converted into rules using the mapping shown in table 6.7. This interpretation is a result of applying all length-two schema to the binary patterns representing votes shown in table 6.4 on page 106. Partial matches are given half credit. In addition, each immune system rule contains a matching threshold that must be exceeded if the rule is to fire. This value is decoded from the B-lymphocyte threshold gene as shown in figure 6.7 on page 102. The conversion of AQ conjunctive descriptions into rules is trivial. The only difference between the AQ and immune system rule representations is that the AQ rules have no explicit thresholds. If an example to be classified does not match any of the rules perfectly, AQ15 uses a combination of the strength of the partial match and the prior probability of the classes to make a decision. See (Michalski, Mozetic, Hong, and Lavrac 1986) for more details concerning AQ15 rule interpretation.

The rules produced by the first run of the immune system model are shown in figure 6.10 on the following page, and the rules produced by the first run of AQ are shown in figure 6.11 on page 111. To enable the roles played by these rules to be further visualized, the number of training set examples covered and classified by each of the rules is shown in figures 6.12 and 6.13 on page 112. By *covered*, we mean the number of examples that would have been classified by the rule if it were the only rule in the set. Although in practice multiple rules typically match each example, both the immune system model and AQ choose a single rule to perform the classification based on some measure of strength; therefore, most of the "covered" bars are much taller than the "classified" bars. In other words, an example can be covered by many rules but will only be classified by a single rule. The exception is the first rule generated by AQ, which always classifies everything it covers.

In analyzing these rule sets, the first observed difference is that, as previously noted, significantly fewer rules were produced by the immune system model than by AQ. Not only are there fewer rules, but the total number of conjuncts in the rule set is smaller—the immune system rule set contains a total of 25 conjuncts while the AQ rule set contains 29

Rule 1:
IF 7 percent **OF**
v4 = abstain or nay
THEN democrat

Rule 2:
IF 67 percent **OF**
v4 = abstain or nay
v6 = yea
v9 = yea
v10 = half credit for yea or nay
v11 = nay
v12 = nay or half credit for abstain
v13 = abstain or yea
v14 = yea or half credit for abstain
v15 = yea
THEN democrat

Rule 3:
IF 54 percent **OF**
v2 = abstain or yea
v4 = nay
v5 = nay
v6 = abstain or half credit for yea or nay
v9 = yea
v11 = yea or half credit for abstain
v13 = abstain or nay
THEN democrat

Rule 4:
IF 77 percent **OF**
v3 = abstain or yea
v7 = abstain or nay
v8 = abstain or nay
v11 = abstain or yea
v12 = nay
v14 = abstain or yea
THEN democrat

Rule 5:
IF 98 percent **OF**
v13 = abstain or yea
v14 = nay
THEN democrat

Figure 6.10: Rule-based interpretation of B-cells from final immune system cover

Rule 1:
IF v4 = abstain or nay
v3 = yea
THEN democrat

Rule 2:
IF v4 = nay
v12 = yea or nay
v6 = yea
THEN democrat

Rule 3:
IF v15 = yea
v14 = yea or nay
v2 = abstain or yea
THEN democrat

Rule 4:
IF v3 = abstain or yea
v11 = abstain or yea
v9 = yea or nay
v7 = abstain or nay
THEN democrat

Rule 5:
IF v3 = yea
v16 = abstain
v13 = yea
THEN democrat

Rule 6:
IF v5 = nay
v15 = yea
v3 = nay
THEN democrat

Rule 7:
IF v13 = nay
v2 = yea
v3 = nay
THEN democrat

Rule 8:
IF v12 = nay
v11 = abstain or yea
v16 = abstain
v3 = abstain or nay
THEN democrat

Rule 9:
IF v11 = yea
v2 = nay
v1 = nay
v16 = nay
THEN democrat

Figure 6.11: Rule-based interpretation of AQ conjunctive descriptions

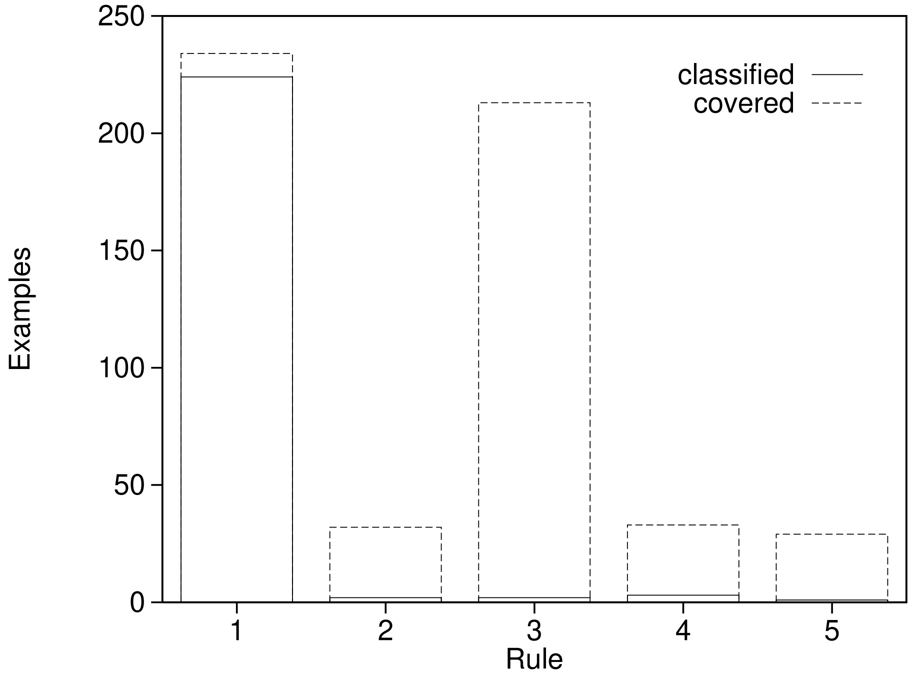


Figure 6.12: Immune system rule coverage and classification

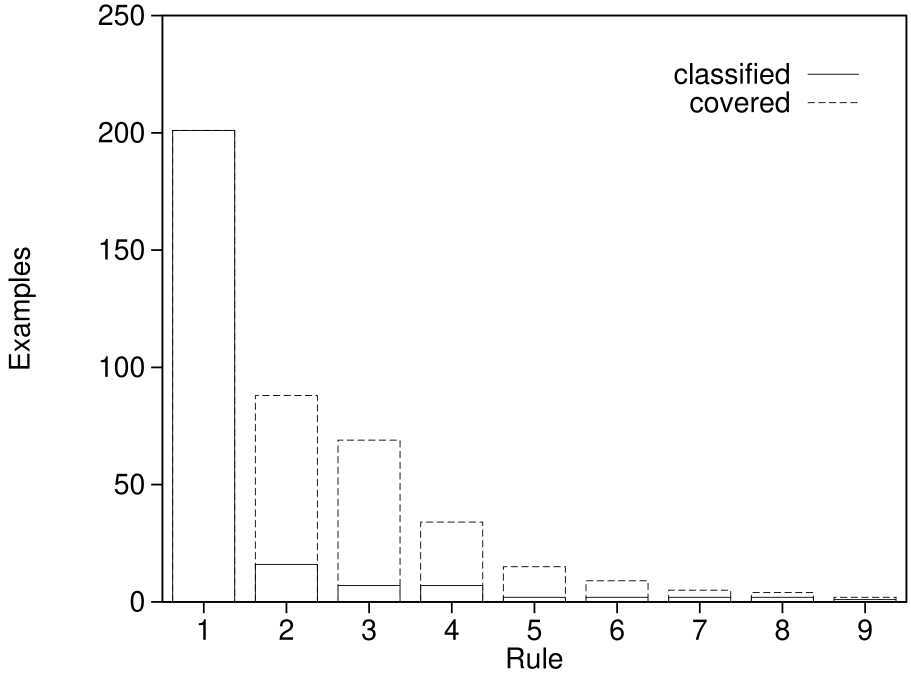


Figure 6.13: AQ rule coverage and classification

conjuncts. Second, the AQ rules are all at about the same level of generality, while the immune system rules vary from very general to quite specific. This second observation is a possible explanation for the smaller number of rules produced by the immune system model. By being more flexible in constructing rules with a wide range of generality, the immune system model is able to discover a more optimal-sized decomposition.

The rule sets also have a number of similar characteristics. First, the initial rules produced by both the immune system and AQ are very similar; specifically, they both consider an abstain or nay on issue number four to be strong evidence that the voting record belongs to a democrat. Furthermore, this is the most general rule produced by both methods. From figures 6.12 and 6.13 one can see that this initial rule classifies most of the examples. In other words, both methods have discovered that the vote on issue number four is the most important discriminator. However, the immune system solution places more emphasis on this discovery than the AQ solution. The second similarity is that the decompositions produced by both methods must rely on a few rules that match only one or two examples to cover the training set adequately.

As in the previous case study, cooperative coevolution has demonstrated its ability to produce good problem decompositions using a task-independent approach in which the number of subcomponents and the role each will play emerges purely as a result of evolutionary pressure. The decomposition produced by the coevolutionary immune system model on the congressional voting records data set is actually better than that produced by AQ15 with respect to the size of the rule set. In addition, AQ15 has the disadvantages of requiring that a set of preclassified training examples is available and of being much more task-specific. The only disadvantage we have found to using the coevolutionary immune system model for concept learning is that it is more computationally expensive than the task-specific symbolic approaches.

6.3 Summary

In summary, through a direct comparison with two task-specific techniques, these relatively difficult case studies have verified the robustness of our task-independent approach in which problem decompositions emerge purely as a result of evolutionary pressure to cooperate. Given problems from the domains of concept learning and neural network construction that are only decomposable into subtasks with complex and difficult to understand interdependencies, the model of cooperative coevolution was able to discover important environmental niches and evolve subcomponents appropriate in number and generality to cover those niches. We emphasize that cooperative coevolution is an extremely general approach to problem decomposition that can be applied to both supervised and reinforcement learning tasks, yet the method produced decompositions as good as or better than those produced by two highly task-specific approaches.

In addition to achieving our primary goal with respect to emergent problem decomposition, the chapter makes two secondary contributions. First, it demonstrates the applicability of the coevolutionary model to evolution strategies as well as to genetic algorithms. We strongly believe the model could be effectively applied to other classes of evolutionary algorithms as well. Further evidence in support of this hypothesis can be found in earlier work on applying cooperative coevolution to the SAMUEL system (Potter, De Jong, and

Grefenstette 1995). Second, it demonstrates the effectiveness of a novel approach to concept learning in which cooperative coevolution is applied to a computational model of one of the recognition processes within the vertebrate immune system. Whether computer simulations of the immune system will join neural networks and evolution as prevalent biologically inspired tools for the solution of technical problems remains to be seen; however, this study represents an initial step towards that purpose.

Chapter 7

CONCLUSIONS

7.1 Summary

This dissertation has addressed a serious limitation of traditional evolutionary algorithms that reduces their effectiveness when applied to increasingly complex problems, namely they lack the explicit notion of modularity required to provide reasonable opportunities for solutions to evolve in the form of interacting coadapted subcomponents. Our goal has been to find computational extensions to the current evolutionary paradigms in which such subcomponents "emerge" rather than being designed by hand. The primary issues have been 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.

To accomplish this mission, we designed and analyzed a novel computational model of cooperative coevolution in which the subcomponents of a problem solution are drawn from a collection of species that interact within a common ecosystem yet are genetically isolated. As described in chapter 3, each individual in the ecosystem is rewarded based on how well it collaborates with individuals from other species to achieve a common goal. The dynamics of the model are such that reasonable problem decompositions emerge due to evolutionary pressure rather than being specified by the user. The model is a general problem-solving method that is applicable to a variety of domains, and is not limited to any particular underlying evolutionary algorithm. The evolution of genetically isolated species in separate populations can be easily distributed across a network of processors with little communication overhead, and unproductive cross-species mating is eliminated through genetic isolation. In addition, by evaluating individuals from one species within the context of individuals from other species, the search space is constrained.

In chapter 4 we performed a sensitivity analysis on some of the primary characteristics of decomposable problems likely to affect the performance of the coevolutionary model. Specifically, we analyzed the importance of the amount and structure of interdependency between problem subcomponents, the dimensionality of the decomposition, and the accuracy of the collaboration fitness evaluations. Regarding the amount and structure of interdependency, the study demonstrated that the performance of the coevolutionary model gracefully declines with an increase in the random epistatic interactions between species. This has positive implications for the application of cooperative coevolution to the solution of a broad class of problems with complex interdependencies between subcomponents. However, when there are many highly structured interactions, as we found in some pathological problems from the domain of real-valued function optimization, the model is quite susceptible to becoming frozen in Nash equilibrium. We hypothesized that further research into alternative strategies for forming collaborations is likely to lead to models of coevolution less susceptible to this difficulty. Results from an analysis of the effect of dimensionality on the coevolutionary model were even more encouraging. The scalability of the model suggests that coevolution may be suitable for the solution of extremely large problems; especially when one considers the potential for parallelizing the model. Regarding the effect of inaccuracy of collaboration fitness evaluations, although the model was less resistant to noise than the standard evolutionary model, we are confident that further research into alternative collaboration strategies will lead to more robust coevolutionary models.

In chapter 5 we explored the basic problem decomposition capability of the model of cooperative coevolution. We demonstrated, within the context of a simple string covering problem, that the model is capable of provoking the emergence of species that work together to cover multiple environmental niches, evolve to an appropriate level of generality, and adapt to a changing environment. It accomplishes this through a task-independent approach in which the problem decomposition emerges purely as a result of evolutionary pressure to cooperate. We also investigated a technique for dynamically creating new species and eliminating unproductive ones. The technique resulted in the emergence of an appropriate number of coadapted species.

Finally, in chapter 6 we applied the model of cooperative coevolution to problems from the domains of concept learning and artificial neural network construction that are only decomposable into subtasks with complex and difficult to understand interdependencies. The resulting problem decompositions were compared and contrasted with those produced by task-specific non-evolutionary methods. These case studies verified the robustness of our task-independent approach in which problem decompositions emerge purely as a result of evolutionary pressure to cooperate. The model of cooperative coevolution was able to discover important environmental niches and evolve subcomponents appropriate in number and generality to cover those niches. Along with achieving the primary goal of validating our approach to emergent problem decomposition, both case studies made secondary contributions. The neural network study demonstrated the applicability of the coevolutionary model to evolution strategies as well as to genetic algorithms. We strongly believe the model could be effectively applied to other classes of evolutionary algorithms as well. The concept learning study demonstrated the effectiveness of a novel approach to machine learning in which cooperative coevolution is applied to a computational model of one of the recognition processes within the vertebrate immune system. Whether computer simulations of the immune system will join neural networks and evolution as prevalent biologically inspired tools for the solution of technical problems remains to be seen; however, this study represents an initial step towards that purpose.

7.2 Future Research

Throughout this dissertation we have suggested a number of possible directions for future research into the design and analysis of computational models of cooperative coevolution. To conclude, we now briefly expand on a number of these ideas.

Alternative Collaboration Strategies

The experiments described in this dissertation used a greedy collaboration strategy in which all the individuals from one species are evaluated within the context of the best individual from each of the other species. We chose this strategy because it is simple and requires a minimal number of collaborations between individuals to be evaluated. However, we showed in chapter 4 that this strategy has some undesirable characteristics. An important area for future research is the study of alternative collaboration strategies. The patterns of interaction, collaborative and otherwise, between interdependent species in nature can be quite complex. One possible research direction would be to turn to the field of ecology for inspiration in designing more biologically faithful collaboration strategies.

Alternative Ecological Relationships

This dissertation focused entirely on the ecological relationship known as mutualism in which each species helps the other. Species in natural ecosystems also have competitive and exploitative relationships. An interesting future research direction would be to apply our basic model of genetically isolated species to the study of these alternative relationships. Some work on coevolving species with competitive relationships has already been done by other researchers; for example, see (Hillis 1991; Rosin and Belew 1995). More advanced studies should model the coevolution of species having a variety of different types of ecological relationships.

Alternative Models of Speciation

When we introduce a new species into an ecosystem, we always initialize its population randomly. However, in nature new species generally arise from existing species. Much more research needs to be done in the design and analysis of more biologically faithful computational models of speciation.

Parallel Implementations

All the experimental studies described in this dissertation have used a sequential single-processor implementation of cooperative coevolution in which each species is evolved in turn for a single generation. However, not only can our model of cooperative coevolution take advantage of all the previous methods for parallelizing evolutionary algorithms, but each species can also be evolved by its own semiautonomous evolutionary algorithm running on its own computer. One advantage of our model with respect to this form of parallelism is that each species can be evolved asynchronously. Another advantage is that little communication between species is required. Some preliminary studies suggest that there are cases in which it may be advantageous to limit communication between species even more than in our current model. There is clearly a need for more research into parallel implementations of cooperative coevolution. We envision ecosystems of hundreds, or even thousands of coadapting species interacting over vast computer networks.

Heterogeneous Representations

One major advantage of our coevolutionary model over previous computational models of evolution is the ease in which one can evolve individuals with heterogeneous representations. As in nature, each species in our model is genetically isolated; therefore, there is no requirement for their chromosomes to be compatible. It is even possible to mix evolutionary algorithms in the same system, as in evolving some species having genotypic representations with genetic algorithms, and others having phenotypic representations with evolution strategies or the evolutionary programming paradigm. The only requirement is the ability for the species to interact with one another. Perhaps some form of common interface between species could be designed to facilitate this process. We feel that the evolution of species with heterogeneous representations is an exciting enabling technology for problem solving in highly complex domains.

Coevolving Complex Behaviors

A possible application area for our coevolutionary model is in learning behaviors for autonomous robots or intelligent agents. In applying our model to behavior learning, each species could represent a different area of expertise. We have already published the results of some preliminary research in which cooperative coevolution was used to develop a rule-based system of behaviors for an autonomous robot (Potter, De Jong, and Grefenstette 1995). This work was an extension of a system called SAMUEL, which was designed to evolve sets of sequential decision rules to be used by decision-making agents (Grefenstette, Ramsey, and Schultz 1990). Although encouraging, for a variety of reasons this work was somewhat inconclusive. However, after more research has been completed in the areas mentioned above it is likely that much more progress could be made in the area of coevolving complex behaviors.

Coevolutionary Models of Molecular Biology

In chapter 6, we used coevolution in conjunction with a loose model of one of the recognition processes within the vertebrate immune system as a concept learning system. One possible research direction would be more investigation into the use of coevolutionary models of the immune system for the solution of technical problems. There is, however, an alternative research direction. Although our focus here has been on emergent problem decomposition, not molecular biology, we believe that great potential lies in collaborating with immunologists to build more biologically faithful coevolutionary models of the immune system to gain insight into the process by which our bodies overcome infection. This may be helpful, for example, in our fight against the human immunodeficiency virus (HIV) or in making headway against the disease of cancer.

BIBLIOGRAPHY

BIBLIOGRAPHY

98th Congress (1984). *Congressional Quarterly Almanac*, Volume XL. Washington, D.C.: Congressional Quarterly Inc.

Ackley, D. H. (1987). *A Connectionist Machine for Genetic Hillclimbing.* Kluwer Academic.

Amdahl, G. M. (1967). Validity of the single-processor approach to achieving large scale computing capabilities. In *AFIPS Conference Proceedings*, Volume 30, pp. 483–485. AFIPS Press.

Axelrod, R. M. (1984). *The Evolution of Cooperation.* New York: Basic Books.

Bäck, T. and H.-P. Schwefel (1993). An overview of evolutionary algorithms for parameter optimization. *Evolutionary Computation 1*(1), 1–23.

Beasley, D., D. R. Bull, and R. R. Martin (1993). A sequential niche technique for multimodal function optimization. *Evolutionary Computation 1*(2), 101–125.

Belew, R. K. (1989). Back propagation for the classifier system. In J. D. Schaffer (Ed.), *Proceedings of the Third International Conference on Genetic Algorithms*, pp. 275–281. Morgan Kaufmann.

Bellman, R. (1957). *Dynamic Programming.* Princeton, New Jersey: Princeton University Press.

Bhattacharyya, G. K. and R. A. Johnson (1977). *Statistical Concepts and Methods.* John Wiley & Sons.

Boag, P. T. and P. R. Grant (1981). Intense natural selection in a population of Darwin's finches (Geospizinae) in the Galápagos. *Science 214*, 82–85.

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

Carroll, L. (1871). *Through the Looking-Glass and What Alice Found There.* London: Macmillan and Co.

Cobb, H. G. (1990). An investigation into the use of hypermutation as an adaptive operator in genetic algorithms having continuous time-dependent nonstationary environments. Technical Report NRL Memorandum 6760, Naval Research Laboratory.

Cohoon, J. P., S. U. Hegde, W. N. Martin, and D. Richards (1987). Punctuated equilibria: A parallel genetic algorithm. In J. J. Grefenstette (Ed.), *Proceedings of the Second International Conference on Genetic Algorithms*, pp. 148–154. Lawrence Erlbaum Associates.

Compiani, M., D. Montanari, R. Serra, and G. Valastro (1988). Classifier systems and neural networks. In E. R. Caianiello (Ed.), *Parallel Architectures and Neural Networks: First Italian Workshop*, pp. 105–118. World Scientific.

Cramer, N. L. (1985). A representation for the adaptive generation of simple sequential programs. In J. J. Grefenstette (Ed.), *Proceedings of an International Conference on Genetic Algorithms and Their Applications*, pp. 183–187. Lawrence Erlbaum Associates.

Darwen, P. (1996). *Co-Evolutionary Learning by Automatic Modularisation with Speciation*. Ph. D. thesis, University of New South Wales, Canberra, Australia.

Darwen, P. and X. Yao (1996). Automatic modularization by speciation. In *Proceedings of the Third IEEE International Conference on Evolutionary Computation*, pp. 88–93. IEEE Press.

Darwin, C. (1859). *On the Origin of Species by Means of Natural Selection*. London: John Murry.

Das, R. and D. Whitley (1991). The only challenging problems are deceptive: Global search by solving order-1 hyperplanes. In R. K. Belew and L. B. Booker (Eds.), *Proceedings of the Fourth International Conference on Genetic Algorithms*, pp. 166–173. Morgan Kaufmann.

Davidor, Y. (1991). A naturally occuring niche & species phenomenon: The model and first results. In R. K. Belew and L. B. Booker (Eds.), *Proceedings of the Fourth International Conference on Genetic Algorithms*, pp. 257–263. Morgan Kaufmann.

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

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

de Garis, H. (1996). "CAM-Brain" ATR's billion neuron artificial brain project. In R. S. Michalski and J. Wnek (Eds.), *Proceedings of the Third International Workshop on Multistrategy Learning*, pp. 251–269. AAAI Press.

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

De Jong, K. A. (1990). Genetic-algorithm-based learning. In Y. Kodratoff and R. S. Michalski (Eds.), *Machine Learning*, Volume 3, pp. 611–638. Morgan Kaufmann.

De Jong, K. A. (1993). Genetic algorithms are not function optimizers. In L. D. Whitley (Ed.), *Foundations of Genetic Algorithms 2*, pp. 5–17. Morgan Kaufmann.

De Jong, K. A., W. M. Spears, and D. F. Gordon (1993). Using genetic algorithms for concept learning. *Machine Learning 13*(2/3), 5–188.

Dixon, L. C. W. (1974). Nonlinear optimization: A survey of the state of the art. In D. J. Evans (Ed.), *Software for Numerical Mathematics*, pp. 193–216. Academic Press.

Doorenbos, R. B. (1994). Combining left and right unlinking for matching a large number of learned rules. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, Volume 1, pp. 451–458. AAAI Press/The MIT Press.

Edwards, S. F. and P. W. Anderson (1975). Theory of spin glasses. *Journal of Physics 5*, 965–974.

Fahlman, S. E. (1988). An empirical study of learning speed in back-propagation networks. Technical Report CMU-CS-88-162, Carnegie Mellon University.

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

Farmer, J. D. (1991). A rosetta stone for connectionism. In S. Forrest (Ed.), *Emergent Computation*, pp. 153–187. The MIT Press.

Farmer, J. D., N. H. Packard, and A. S. Perelson (1986). The immune system, adaptation and machine learning. *Physica D 22*, 187–204.

Fitzpatrick, J. M. and J. J. Grefenstette (1988). Genetic algorithms in noisy envoronments. *Machine Learning 3*, 101–120.

Fogel, L. J., A. J. Owens, and M. J. Walsh (1966). *Artificial Intelligence Through Simulated Evolution.* John Wiley & Sons.

Forgy, C. L. (1982). Rete: A fast algorithm for the many pattern/many object pattern match problem. *Artificial Intelligence 19*(2), 17–37.

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

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

Friedman, M. and L. S. Savage (1947). Planning experiments seeking maxima. In C. Eisenhart, M. W. Hastay, and W. A. Wallis (Eds.), *Selected Techniques of Statistical Analysis for Scientific and Industrial Research, and Production and Management Engineering*, pp. 363–372. New York: McGraw-Hill Book Co.

Fujiki, C. and J. Dickinson (1987). Using the genetic algorithm to generate lisp source code to solve the prisoner's dilemma. In J. J. Grefenstette (Ed.), *Proceedings of the Second International Conference on Genetic Algorithms*, pp. 236–240. Lawrence Erlbaum Associates.

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

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

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

Goldberg, D. E., B. Korb, and K. Deb (1989). Messy genetic algorithms: Motivation, analysis, and first results. *Complex Systems 3*(5), 493–530.

Goldberg, D. E. and J. Richardson (1987). Genetic algorithms with sharing for multimodal function optimization. In J. J. Grefenstette (Ed.), *Proceedings of the Second International Conference on Genetic Algorithms*, pp. 41–49. Lawrence Erlbaum Associates.

Goldberg, D. E. and R. E. Smith (1987). Nonstationary function optimization using genetic algorithms with dominance and diploidy. In J. J. Grefenstette (Ed.), *Proceedings of the Second International Conference on Genetic Algorithms*, pp. 59–68. Lawrence Erlbaum Associates.

Gordon, V. S. and D. Whitley (1993). Serial and parallel genetic algorithms as function optimizers. In S. Forrest (Ed.), *Proceedings of the Fifth International Conference on Genetic Algorithms*, pp. 177–183. Morgan Kaufmann.

Gorges-Schleuter, M. (1989). ASPARAGOS an asynchronous parallel genetic optimization strategy. In J. D. Schaffer (Ed.), *Proceedings of the Third International Conference on Genetic Algorithms*, pp. 422–427. Morgan Kaufmann.

Grefenstette, J. J. (1989). A system for learning control strategies with genetic algorithms. In J. D. Schaffer (Ed.), *Proceedings of the Third International Conference on Genetic Algorithms*, pp. 183–190. Morgan Kaufmann.

Grefenstette, J. J. (1992). Genetic algorithms for changing environments. In R. Männer and B. Manderick (Eds.), *Parallel Problem Solving from Nature, 2*, pp. 137–144. Elsevier Science.

Grefenstette, J. J. and J. M. Fitzpatrick (1985). Genetic search with approximate function evaluations. In J. J. Grefenstette (Ed.), *Proceedings of an International Conference on Genetic Algorithms and Their Applications*, pp. 112–120. Lawrence Erlbaum Associates.

Grefenstette, J. J., C. L. Ramsey, and A. C. Schultz (1990). Learning sequential decision rules using simulation models and competition. *Machine Learning 5*(4), 355–381.

Grosso, P. B. (1985). *Computer Simulations of Genetic Adaptation: Parallel Subcomponent Interaction in a Multilocus Model.* Ph. D. thesis, University of Michigan, Ann Arbor, MI.

Hadley, G. (1964). *Nonlinear and Dynamic Programming.* Reading, Mass.: Adison-Wesley.

Hamilton, W. D. (1982). Pathogens as causes of genetic diversity in their host. In R. M. Anderson and R. M. May (Eds.), *Population Biology of Infectious Diseases*, pp. 269–296. Springer-Verlag.

Hicklin, J. F. (1986). Application of the genetic algorithm to automatic program generation. Master's thesis, Department of Computer Science, University of Idaho.

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

Holland, J. H. (1975). *Adaptation in Natural and Artificial Systems.* University of Michigan Press.

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

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

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

Janikow, C. Z. (1991). *Inductive Learning from Attribute-Based Examples: A Knowledge-Intensive Genetic Algorithm Approach.* Ph. D. thesis, University of North Carolina at Chapel Hill.

Janikow, C. Z. (1993). A knowledge-intensive genetic algorithm for supervised learning. *Machine Learning 13*(2/3), 189–228.

Jones, T. (1995). *Evolutionary Algorithms, Fitness Landscapes, and Search.* Ph. D. thesis, University of New Mexico, Albuquerque, NM.

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

Kauffman, S. A. (1989). Adaptation on rugged fitness landscapes. In D. L. Stein (Ed.), *Lectures in the Sciences of Complexity*, Volume 1, pp. 527–618. Addison Wesley.

Kauffman, S. A. (1993). *The Origins of Order.* Oxford University Press.

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

Kettlewell, H. B. D. (1955). Selection experiments on industrial melanism in the lepidoptera. *Heredity 9*, 323–342.

Koza, J. R. (1989). Hierarchical genetic algorithms operating on populations of computer programs. In N. S. Sridharan (Ed.), *Eleventh International Joint Conference on Artificial Intelligence*, pp. 768–774. Morgan Kaufmann.

Koza, J. R. (1992). *Genetic Programming.* The MIT Press.

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

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

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

Lenat, D. B. (1995). CYC: a large-scale investment in knowledge infrastructure. *Communications of the ACM 38*(11), 33–38.

Lewis, T. G. and E.-R. Hesham (1992). *Introduction to Parallel Computing*. Prentice-Hall.

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

Manderick, B. and P. Spiessens (1989). Fine-grained parallel genetic algorithms. In J. D. Schaffer (Ed.), *Proceedings of the Third International Conference on Genetic Algorithms*, pp. 428–433. Morgan Kaufmann.

McInerney, J. (1992). *Biologically Influenced Algorithms and Parallelism in Non-linear Optimization*. Ph. D. thesis, University of California, San Diego, La Jolla, CA.

Michalski, R. S. (1983). A theory and methodology of inductive learning. In R. S. Michalski, J. G. Carbonell, and T. M. Mitchell (Eds.), *Machine Learning*, pp. 83–134. Morgan Kaufmann.

Michalski, R. S., I. Mozetic, J. Hong, and N. Lavrac (1986). The AQ15 inductive learning system: An overview and experiments. Technical Report UIUCDCS-R-86-1260, University of Illinois, Urbana-Champaign, IL.

Miller, G. F., P. M. Todd, and S. U. Hegde (1989). Designing neural networks using genetic algorithms. In J. D. Schaffer (Ed.), *Proceedings of the Third International Conference on Genetic Algorithms*, pp. 379–384. Morgan Kaufmann.

Miller, R. G. (1986). *Beyond ANOVA, basics of applied statistics*. John Wiley & Sons.

Montana, D. J. and L. Davis (1989). Training feedforward neural networks using genetic algorithms. In N. S. Sridharan (Ed.), *Eleventh International Joint Conference on Artificial Intelligence*, pp. 762–767. Morgan Kaufmann.

Moriarty, D. E. and R. Miikkulainen (1996). Efficient reinforcement learning through symbiotic evolution. *Machine Learning 22*(1), 11–33.

Mühlenbein, H. (1989). Parallel genetic algorithms, population genetics and combinatorial optimization. In J. D. Schaffer (Ed.), *Proceedings of the Third International Conference on Genetic Algorithms*, pp. 416–421. Morgan Kaufmann.

Nash, J. (1951). Non-cooperative games. *Annals of Mathematics 54*(2), 286–295.

Neri, F. and L. Saitta (1996). Exploring the power of genetic search in learning symbolic classifiers. To appear in IEEE Transactions on Pattern Analysis and Machine Intelligence.

Oren, S. S. (1974). On the selection of parameters in self scaling variable metric algorithms. *Mathematical Programming 7*, 351–367.

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

Perry, Z. (1984). *Experimental Study of Speciation in Ecological Niche Theory Using Genetic Algorithms*. Ph. D. thesis, University of Michigan, Ann Arbor, MI.

Pettey, C. B., M. R. Leuze, and J. J. Grefenstette (1987). A parallel genetic algorithm. In J. J. Grefenstette (Ed.), *Proceedings of the Second International Conference on Genetic Algorithms*, pp. 155–161. Lawrence Erlbaum Associates.

Pettit, E. and K. M. Swigger (1983). An analysis of genetic-based pattern tracking and cognitive-based component tracking models of adaptation. In *Proceedings of the National Conference on Artificial Intelligence (AAAI-83)*, pp. 327–332. William Kaufmann, Inc.

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

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

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

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

Quinlan, J. R. (1986). Induction of decision trees. *Machine Learning 1*, 81–106.

Rastrigin, L. A. (1974). *Extremal Control Systems*. Moscow: Nauka. Theoretical Foundations of Engineering Cybernetics Series (in Russian).

Rechenberg, I. (1964). Cybernetic solution path of an experimental problem. Library Translation 1122, August 1965. Farnborough Hants: Royal Aircraft Establishment. English translation of lecture given at the Annual Conference of the WGLR at Berlin in September, 1964.

Rechenberg, I. (1973). *Evolutionsstrategie—Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. Stuttgart-Bad Cannstatt: Frommann-Holzboog.

Roitt, I. M. (1994). *Essential Immunology* (Eighth ed.). Blackwell Scientific Publications.

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

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

Rosenbrock, H. H. (1960). An automatic method for finding the greatest or least value of a function. *Computer Journal 3*, 175–184.

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

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

Salomon, R. (1996). Reevaluating genetic algorithm performance under coordinate rotation of benchmark functions. *BioSystems 39*, 263–278.

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

Schaffer, J. D., D. Whitley, and L. J. Eshelman (1992). Combinations of genetic algorithms and neural networks: A survey of the state of the art. In L. D. Whitley and J. D. Schaffer (Eds.), *COGANN-92 International Workshop on Combinations of Genetic Algorithms and Neural Networks*, pp. 1–37. IEEE Computer Society Press.

Schlimmer, J. C. (1987). *Concept Acquisition through Representational Adjustment*. Ph. D. thesis, University of California, Irvine, CA.

Schwefel, H.-P. (1981). *Numerical Optimization of Computer Models*. John Wiley & Sons. English translation of Numerische Optimierung von Computer-Modellen mittels der Evolutionsstrategie, 1977.

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

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*. D. Appleton-Century, New York.

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

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

Smith, S. F. (1983). Flexible learning of problem solving heuristics through adaptive search. In A. Bundy (Ed.), *Proceedings of the Eighth International Joint Conference on Artificial Intelligence*, pp. 422–425. William Kaufmann.

Southwell, R. V. (1946). *Relaxation Methods in Theoretical Physics*. Oxford UK: Clarendon Press.

Spears, W. M. (1994). Simple subpopulation schemes. In A. V. Sebald and D. B. Fogel (Eds.), *Proceedings of the Third Conference on Evolutionary Programming*, pp. 297–307. World Scientific.

Spedicato, E. (1975). Computational experience with quasi-Newton algorithms for minimization problems of moderately large size. Technical Report CISE-N-175, Centro Informazioni Studi Esperienze, Segrate (Milano), Italy.

Spiessens, P. and B. Manderick (1991). A massively parallel genetic algorithm implementation and first analysis. In R. K. Belew and L. B. Booker (Eds.), *Proceedings of the Fourth International Conference on Genetic Algorithms*, pp. 279–286. Morgan Kaufmann.

Spofford, J. J. and K. J. Hintz (1991). Evolving sequential machines in amorphous neural netrorks. In T. Kohonen, K. Mäkisara, O. Simula, and J. Kangas (Eds.), *Artificial Neural Networks*, pp. 973–978. Elsevier Science.

Stadnyk, I. (1987). Schema recombination in a pattern recognition problem. In J. J. Grefenstette (Ed.), *Proceedings of the Second International Conference on Genetic Algorithms*, pp. 27–35. Lawrence Erlbaum Associates.

Steele, Jr., G. L. (1990). *Common Lisp the Language* (Second ed.). Woburn, MA: Digital Press.

Student (1908). The probable error of a mean. *Biometrika 6*, 1–25.

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

Tanese, R. (1987). Parallel genetic algorithm for a hypercube. In J. J. Grefenstette (Ed.), *Proceedings of the Second International Conference on Genetic Algorithms*, pp. 177–183. Lawrence Erlbaum Associates.

Tanese, R. (1989). Distributed genetic algorithms. In J. D. Schaffer (Ed.), *Proceedings of the Third International Conference on Genetic Algorithms*, pp. 434–439. Morgan Kaufmann.

Thrun, S. B., J. Bala, E. Bloedorn, I. Bratko, B. Cestnik, J. Cheng, K. De Jong, S. Džeroski, S. E. Fahlman, D. Fisher, R. Hamann, K. Kaufman, S. Keller, I. Kononenko, J. Kreuziger, R. S. Michalski, T. Mitchell, P. Pachowicz, Y. Reich, H. Vafaie, W. Van de Welde, W. Wenzel, J. Wnek, and J. Zhang (1991). The MONK's problems—a performance comparison of different learning algorithms. Technical Report CMU-CS-91-197, Carnegie Mellon University.

Turner, J. R. G. (1977). Butterfly mimicay: the genetical evolution of an adaption. *Evolutionary Biology 10*, 163–206.

Van Valen, L. (1973). A new evolutionary law. *Evolutionary Theory 1*, 1–30.

Watkins, C. J. C. H. (1989). *Learning from Delayed Rewards*. Ph. D. thesis, University of Cambridge, England.

Watkins, C. J. C. H. and P. Dayan (1992). $\mathcal{Q}$-learning. *Machine Learning 8*, 279–292.

Weiss, S. M. and C. A. Kulikowski (1991). *Computer Systems that Learn*. Morgan Kaufmann.

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

Whitley, D., K. Mathias, S. Rana, and J. Dzubera (1995). Building better test functions. In L. Eshelman (Ed.), *Proceedings of the Sixth International Conference on Genetic Algorithms*, pp. 239–246. Morgan Kaufmann.

Whitley, D. and T. Starkweather (1990). Genitor II: a distributed genetic algorithm. *Journal of Experimental and Theoretical Artificial Intelligence 2*, 189–214.

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

Wolpert, D. H. and W. G. Macready (1995). No free-lunch theorems for search. Technical Report 95-02-010, Santa Fe Institute.

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

APPENDICES

Appendix A

PROGRAM CODE FOR COOPERATIVE COEVOLUTION MODEL

All the programs used in the experimental studies discussed in this dissertation were written in the computer language Common Lisp (Steele 1990). To enable the reader to resolve any ambiguities in the description of our computational model of cooperation coevolution, we document in this appendix Lisp code that completely implements the example from the final section of chapter 3. The underlying evolutionary algorithm used in this implementation is a genetic algorithm.

Recall from the end of chapter 3 that the coevolutionary model is used to solve a string covering problem in which we are given a set of binary strings with the goal of finding the best possible set of matching strings. These sets are called the *target set* and the *match set* respectively. The number of elements in the match set is equal to the number of species being evolved; that is, each species contributes a single match set element. To make the problem more interesting, we evolve fewer species than strings in the target set so that good solutions are required to contain generalizations rather than simply clones of the target strings. For more background information on the string covering problem and our implementation, see the explanation in section 3.4 beginning on page 39.

Auxiliary Files

Two auxiliary files are required by this Common Lisp implementation. The first auxiliary file is called "targets" and contains the elements of the target set. Each element is represented as a Common Lisp bit vector. The file used in the experiment described in chapter 3 contained the following 32-bit vectors:

```
#*00000000000000000000000000000000
#*00010001000100010001000100010001
#*00110011001100110011001100110011
#*01000100010001000100010001000100
#*01010101010101010101010101010101
#*10001000100010001000100010001000
```

The second auxiliary file is called "randomstates" and contains precomputed Common Lisp random state objects. These objects are used by the Common Lisp pseudo-random number generator to encapsulate state information. By precomputing random state objects using the Common Lisp function `(make-random-state)`, saving them to a file, and using them to initialize the random number generator at the beginning of each experiment, one can execute a series of stochastic runs that are repeatable.

Parameters

The operation of the coevolutionary model can be modified by adjusting the following parameters:

```
(defparameter *maxgen* 100
  "Maximum number of generations to evolve")

(defparameter *goal* 32.0
  "Fitness that will be considered a success and halt evolution")

(defparameter *popsize* 50
  "Size of the population (must be an even number)")

(defparameter *chrom-length* 32
  "Number of bits in a chromosome")

(defparameter *initial-species* 3
  "The number of species that initially exist")

(defparameter *xover-type* :TWOPT
  "The crossover type (either :TWOPT or :UNIFORM)")

(defparameter *crossprob*  0.6
  "Probability of crossover")

(defparameter *mutateprob* (/ 1.0 *chrom-length*)
  "Probability of mutation")
```

Global Variables

All the global variables used in the model of cooperative coevolution are documented below.

```
(defvar *gen* 0
  "The current generation number")

(defvar *ecosystem-gen* 0
  "The number of completed evolutionary cycles through all species")

(defvar *experiment* 1
  "The current experiment number")

(defvar *species* nil
  "A list of species record structures")

(defvar *current-species* nil
  "The species currently being evolved")
```

```
(defvar *last-species-id* 0
  "The identification number of the last species created")

(defvar *newpop* nil
  "A new population of genotypes created from current population")

(defvar *best-individual* .0
  "Index of the best individual in the current population")

(defvar *best-fitness* .0
  "Fitness of the best individual in the current population")

(defvar *worst-fitness* .0
  "Fitness of the worst individual in the current population")

(defvar *average-fitness* .0
  "Average fitness of all individuals in current population")

(defvar *seed* 1
  "Index into a file of random states")

(defvar *target-set* nil
  "A list of binary target strings")
```

Record Structure Definitions

The following record structure represents a species:

```
(defstruct species
  (id (incf *last-species-id*))
  (genotypes (make-array *popsize* :element-type 'bit-vector))
  (fitnesses (make-array *popsize* :element-type 'float))
  (best-fitness 0.0)
  (rep nil))
```

Top Level Routine

Following is the function implementing the top level control loop for the coevolutionary model. This is the function that is executed by the user to start the coevolutionary process. The function takes two arguments. The first argument, `experiments`, specifies the number of experiments that should be run; and the second argument, `seed`, is an index into a file of precomputed random state objects used to initialize the pseudo-random number generator as previously described in the section on auxiliary files. If multiple experiments are run, the first experiment will use the random state object indexed by `seed`, the second experiment will use the random state object indexed by `seed` + 1, and so on. For example, if 20 experiments had previously been run using the default seed of 1 and the user then wanted to repeat just the eighth and ninth experiments, the Lisp form `(run-coevolution 2 8)` would be executed.

```
(defun run-coevolution (&optional (experiments 1) (seed 1))
  "Execute the model of cooperative coevolution"
  ;;
  ;; Initialize the coevolutionary model
  ;;
  (setq *seed* seed
        *experiment* 1)
  (init-model)
  ;;
  ;; Top level control loop
  ;;
  (loop
   (block ecosystem-gen
     (dolist (*current-species* *species*)
       (when (or (>= *gen* *maxgen*)
                 (>= *best-fitness* *goal*))
         ;;
         ;; Experiment complete
         ;;
         (dump-reps)
         (cond ((< *experiment* experiments)
                ;;
                ;; Start up the next experiment
                ;;
                (incf *experiment*)
                (init-model)
                (return-from ecosystem-gen))
               (t
                ;;
                ;; HALT - no more experiments
                ;;
                (return-from run-coevolution))))
       ;;
       ;; Evolve the current species for one generation
       ;;
       (evolve-species))
     (incf *ecosystem-gen*))))
```

Initialization Routines

Initialization of the coevolutionary model is handled by two routines. The first of these routines, `init-model`, is executed at the beginning of each experiment, and the second, `init-species`, is executed each time a new species needs to be created.

```
(defun init-model ()
  "Initialize the coevolutionary model for a new experiment"
  (setq *gen* 0
        *ecosystem-gen* 0
        *species* nil
        *last-species-id* -1
```

```
        *newpop* (make-array *popsize* :element-type 'bit-vector)
        *target-set* nil)
  ;;
  ;; Initialize pseudo-random number generator
  ;;
  (let (randomState)
    (with-open-file (ifile "randomstates" :direction :input)
      (dotimes (i *seed*)
        (setq randomState (read ifile nil nil))))
    (if (random-state-p randomState)
        (setq *random-state* randomState)
        ;;
        ;; Fatal error --- bad random state
        ;;
        (error "Bad random state read from file:  randomstates"))
    (incf *seed*))
  ;;
  ;; Read in the target set
  ;;
  (with-open-file (ifile "targets" :direction :input)
    (let (target)
      (loop
       (if (setq target (read ifile nil nil))
           (setq *target-set* (cons target *target-set*))
           (return)))))
  (setq *target-set* (reverse *target-set*))
  ;;
  ;; Initialize species
  ;;
  (dotimes (i *initial-species*)
    (init-species)))

(defun init-species ()
  "Create and initialize a new species"
  (setq *current-species* (make-species))
  (setq *species* (nconc *species* (list *current-species*)))
  (let ((genotypes (species-genotypes *current-species*))
        chromosome)
    ;;
    ;; Randomly initialize the population
    ;;
    (dotimes (i *popsize*)
      (setq chromosome (make-array *chrom-length* :element-type 'bit))
      (dotimes (j *chrom-length*)
        (setf (aref chromosome j) (random 2)))
      (setf (aref genotypes i) chromosome)))
  (compute-fitness)
  (dump-info)
  (scale-fitness)
  (incf *gen*))
```

Evolutionary Cycle

The following routine implements the select, recombine, evaluate, and replace cycle of a single species. Each pass through this cycle is referred to as a *generation*. Given that this is a sequential implementation, each species is evolved in turn by the routine documented here. When a species is being actively evolved, it is designated the "current species". In contrast, a parallel implementation would evolve all the species simultaneously and there would be no notion of a current species.

```
(defun evolve-species ()
  "Evolve the current species for a single generation"
  (do ((i 0 (+ i 2))
       parent1
       parent2)
      ((= i *popsize*))
    ;;
    ;; Select two individuals to reproduce based on fitness
    ;;
    (setq parent1 (select-parent)
          parent2 (select-parent))
    ;;
    ;; Create offspring through crossover or cloning and
    ;; add them to new population
    ;;
    (multiple-value-bind (child1 child2)
        (recombination parent1 parent2)
      (setf (aref *newpop* i) child1
            (aref *newpop* (1+ i)) child2)))
  ;;
  ;; Mutate the new population if required
  ;;
  (unless (= *mutateprob* 0.0)
    (mutate))
  ;;
  ;; Copy the best individual from the previous generation into
  ;; the new population without modification (elitist strategy)
  ;;
  (setf (aref *newpop* 0)
        (copy-seq (species-rep *current-species*)))
  ;;
  ;; Replace the old population with the new population
  ;;
  (psetf (species-genotypes *current-species*) *newpop*
         *newpop* (species-genotypes *current-species*))
  ;;
  ;; Update the fitnesses and report the status
  ;;
  (compute-fitness)
  (dump-info)
  (scale-fitness)
  (incf *gen*))
```

Selection

The following routine implements fitness proportionate selection. The algorithm behind this routine samples individuals uniformly from the population but only accepts them with probability f_i/f_{max}. This is equivalent, yet usually more efficient in practice, than sampling directly from a fitness proportionate distribution. Specifically, it will only be less efficient when the population contains a small number of individuals with fitness values well above the others.

```
(defun select-parent ()
  "Select individual from the old population based on fitness"
  (let ((fitnesses (species-fitnesses *current-species*))
        (best-fitness (species-best-fitness *current-species*)))
    (do ((sample-index (random *popsize*) (random *popsize*)))
        ((<= (random 1.0) (/ (aref fitnesses sample-index)
                             best-fitness))
         (aref (species-genotypes *current-species*)
               sample-index)))))
```

Genetic Operators

We implement four genetic operators: cloning, two-point crossover, uniform crossover, and bit-flipping mutation. The two crossover operators and cloning are implemented in the routine `recombination`. Cloning occurs implicitly if neither crossover operator is performed. The mutation operator is implemented in the routine `mutate` and uses a geometric distribution to determine which bit in the population to mutate next. That is, the mutation operator treats the entire population of binary genotypes as one long sequence of ones and zeros.

```
(defun recombination (parent1 parent2)
  "Create two children through crossover or cloning"
  (let ((chrom1 (copy-seq parent1))
        (chrom2 (copy-seq parent2))
        cut1
        cut2)
    (when (<= (random 1.0) *crossprob*)
      (cond ((eq *xover-type* :TWOPT)
             ;;
             ;; Two-point crossover
             ;;
             (setq cut1 (random *chrom-length*)
                   cut2 (+ 1 cut1 (random (- *chrom-length* cut1))))
             (psetf
                (subseq chrom1 cut1 cut2) (subseq chrom2 cut1 cut2)
                (subseq chrom2 cut1 cut2) (subseq chrom1 cut1 cut2)))
            (t
             ;;
             ;; Uniform crossover
             ;;
```

```
          (dotimes (i *chrom-length*)
            (when (<= (random 1.0) 0.5)
              (psetf (aref chrom1 i) (aref chrom2 i)
                     (aref chrom2 i) (aref chrom1 i)))))))
    (values chrom1 chrom2)))

(defun mutate ()
  "Mutate the new population"
  (let ((locus 0)
        i
        j)
    (loop
     (setq locus (+ locus (floor (/ (log (random 1.0))
                                    (log (- 1 *mutateprob*)))))
           i (truncate (/ locus *chrom-length*))
           j (mod locus *chrom-length*))
     (when (>= i *popsize*)
       (return))
     (setf (aref (aref *newpop* i) j)
           (if (zerop (aref (aref *newpop* i) j))
               1
               0)))))
```

Fitness Computation

Three functions are used in this implementation to evaluate the fitness of individuals. The first function, `compute-fitness`, implements the control loop for computing the fitness of every individual in the current species. It is also where the species representative is chosen. Although `compute-fitness` is problem-independent, the other two functions in the suite are specific to the string covering problem. The second function, `fitness`, evaluates a single individual based on how well it collaborates with the representatives from the other species to cover the target set. The third function, `match-strength`, simply compares a vector from the match set and a vector from the target set, and returns the number of bits in the same position with the same value.

```
(defun compute-fitness ()
  "Compute the fitness of all individuals in the current species"
  (let ((total-fitness 0.0)
        (fitnesses (species-fitnesses *current-species*))
        (genotypes (species-genotypes *current-species*))
        current-fitness)
    (setq *best-fitness* most-negative-single-float
          *worst-fitness* most-positive-single-float)
    ;;
    ;; Loop through all individuals and compute their fitness
    ;;
    (dotimes (i *popsize*)
      (setq current-fitness (fitness (aref genotypes i)))
      (setf (aref fitnesses i) current-fitness)
      (incf total-fitness current-fitness)
```

```
        (when (> current-fitness *best-fitness*)
          (setq *best-fitness* current-fitness
                *best-individual* i))
        (when (< current-fitness *worst-fitness*)
          (setq *worst-fitness* current-fitness)))
      (setq *average-fitness* (/ total-fitness *popsize*))
      ;;
      ;; Update the current species representative
      ;;
      (setf (species-rep *current-species*)
            (copy-seq (aref genotypes *best-individual*)))))

  (defun fitness (individual)
    "Return the fitness of an individual"
    (let (max-strength
          (strength 0))
      (dolist (target *target-set*)
        ;;
        ;; Find the best match between target string and members of
        ;; collaboration
        ;;
        (setq max-strength 0)
        (dolist (s *species*)
          (setq max-strength
                (max max-strength
                     (if (= (species-id s)
                            (species-id *current-species*))
                         (match-strength individual target)
                         (match-strength (species-rep s) target)))))
        (incf strength max-strength))
      ;;
      ;; Return the average of the best matches
      ;;
      (/ (float strength) (length *target-set*))))

  (defun match-strength (vec1 vec2)
    "Return the similarity between vec1 and vec2"
    (let ((score 0))
      (dotimes (k *chrom-length*)
        (when (= (aref vec1 k) (aref vec2 k))
          (incf score)))
      score))
```

Fitness Scaling

The following function implements balanced linear scaling. In this novel fitness scaling algorithm, the fitness of the average individual will be set to 1.0, the fitness of better than average individuals will be linearly scaled from 1.0 to 2.0, and the fitness of worse than average individuals will be linearly scaled from 0.0 to 1.0.

```lisp
(defun scale-fitness ()
  "Scale the fitness of all individuals in the population"
  (let ((fitnesses (species-fitnesses *current-species*)))
    (cond ((< (- *best-fitness* *worst-fitness*) 0.0001)
           ;;
           ;; All individuals in the population have close to the same
           ;; fitness, so give everyone an equal chance of selection
           ;;
           (dotimes (i *popsize*)
             (setf (aref fitnesses i) 1.0)))
          (t
           ;;
           ;; Scale all fitness values using balanced linear scaling
           ;;
           (let ((m1 (/ 1 (- *best-fitness* *average-fitness*)))
                 (b1 (- 1 (/ *average-fitness*
                             (- *best-fitness* *average-fitness*))))
                 (m2 (/ 1 (- *average-fitness* *worst-fitness*)))
                 (b2 (- 1 (/ *average-fitness*
                             (- *average-fitness* *worst-fitness*)))))
             (dotimes (i *popsize*)
               (cond ((>= (aref fitnesses i) *average-fitness*)
                      ;;
                      ;; This individual is average or above average
                      ;;
                      (setf (aref fitnesses i)
                            (+ (* m1 (aref fitnesses i)) b1)))
                     (t
                      ;;
                      ;; This individual is below average
                      ;;
                      (setf (aref fitnesses i)
                            (+ (* m2 (aref fitnesses i)) b2)))))))))
    ;;
    ;; Update best fitness of species
    ;;
    (setf (species-best-fitness *current-species*)
          (aref fitnesses *best-individual*))))
```

Monitoring Routines

A suite of three routines is used to monitor the progress of the coevolutionary model. The function `dump-info` outputs some initial information about parameter settings and then proceeds to output the status of the evolutionary process at the end of each generation. The function `dump-reps` outputs the representatives of each species, that is, the best match set evolved so far. Finally, the routine `contributions` determines the percentage each species contributed to a collaboration. As discussed in chapter 3, the credit assignment information computed by `contributions` is not used by the coevolutionary process but was required to generate figure 3.6 on page 42.

```
(defun dump-info ()
  "Dump information about current evolutionary status to stdout"
  (when (zerop *gen*)
    ;;
    ;; Output info about the parameters used in this experiment
    ;;
    (format t "STRING MATCHING PROBLEM: ")
    (multiple-value-bind (sec min hour day month year)
        (get-decoded-time)
      (format t " ~3S ~1D, ~4D  ~2,'0D:~2,'0D:~2,'0D~%~%"
              (nth (1- month) '(Jan Feb Mar Apr May Jun Jul
                                Aug Sep Oct Nov Dec))
              day year hour min sec))
    (format t "Seed:          ~10D~%" (1- *seed*))
    (format t "Species:       ~10D~%" *initial-species*)
    (format t "Max Gen:       ~10D~%" *maxgen*)
    (format t "Pop Size:      ~10D~%" *popsize*)
    (format t "Chrom Length:  ~10D~%" *chrom-length*)
    (format t "Xover Prob:    ~10,4F~%" *crossprob*)
    (format t "Mutation Prob: ~10,4F~%" *mutateprob*)
    (format t "Xover Type:    ~10@A~2%" *xover-type*))
  ;;
  ;; Output info about the current generation
  ;;
  (format t "Gen: ~4D  Species: ~2D  Best: ~,2F  Avg: ~,2F"
          *gen* (species-id *current-species*)
          *best-fitness* *average-fitness*)
  ;;
  ;; Output the contribution of each species
  ;;
  (let ((contribs (contributions))
        (index 0))
    (dolist (s *species*)
      (format t "  C~1D: ~,2F"
              (species-id s) (/ (nth index contribs) *best-fitness*))
      (incf index)))
  (terpri))

(defun dump-reps ()
  "Dump species representatives to stdout"
  (terpri)
  (dolist (s *species*)
    (format t "Species~1D: " (species-id s))
    (dotimes (j *chrom-length*)
      (format t "~1D" (aref (species-rep s) j)))
    (terpri)))

(defun contributions ()
  "Compute percentage contribution of each species"
  (let (max-strength
        strength
```

```
    winners
    index
    (contribs (make-list (length *species*)
                         :initial-element 0.0)))
(dolist (target *target-set*)
  ;;
  ;; Find out which species matched target string best and add
  ;; match strength to its contribution.  Break ties randomly.
  ;;
  (setq max-strength -1
        index 0)
  (dolist (s *species*)
    (setq strength (match-strength (species-rep s) target))
    (cond ((> strength max-strength)
           (setq winners (list index)
                 max-strength strength))
          ((= strength max-strength)
           (setq winners (cons index winners))))
    (incf index))
  (incf (nth (nth (random (length winners)) winners) contribs)
        (* 100 (/ (float max-strength) (length *target-set*)))))
;;
;; Return list of contributions in the form of percentages
;;
contribs))
```

Appendix B

PARAMETER OPTIMIZATION PROBLEMS

This appendix contains plots of two-dimensional versions of all the real-valued parameter optimization problems from chapter 4. In this appendix, the functions have all been inverted to provide a better view of the fitness landscape in the vicinity of the global minimum.

Ackley Function

This function was originally proposed by Ackley (1987) and later generalized by Bäck and Schwefel (1993). At a low resolution the landscape of the Ackley function is unimodal; however, the second exponential term covers the landscape with a lattice of many small peaks and basins.

Objective function:

$$f(\vec{x}) = -20 \exp\left(-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}\right) - \exp\left(\frac{1}{n}\sum_{i=1}^{n}\cos\left(2\pi x_i\right)\right) + 20 + e$$

Constraints: $-30.0 \leq x_i \leq 30.0$

Minimum: $\vec{x} = (0, 0, \cdots), \quad f(\vec{x}) = 0.0$

Rastrigin Function

This is a generalized version of a function proposed by Rastrigin (1974). The function is predominantly unimodal with an overlying lattice of moderate sized peaks and basins.

Objective function:

$$f(\vec{x}) = nA + \sum_{i=1}^{n} x_i^2 - A\cos(2\pi x_i)$$

Constraints: $A = 3, \; -5.12 \leq x_i \leq 5.12$

Minimum: $\vec{x} = (0, 0, \cdots), \quad f(\vec{x}) = 0.0$

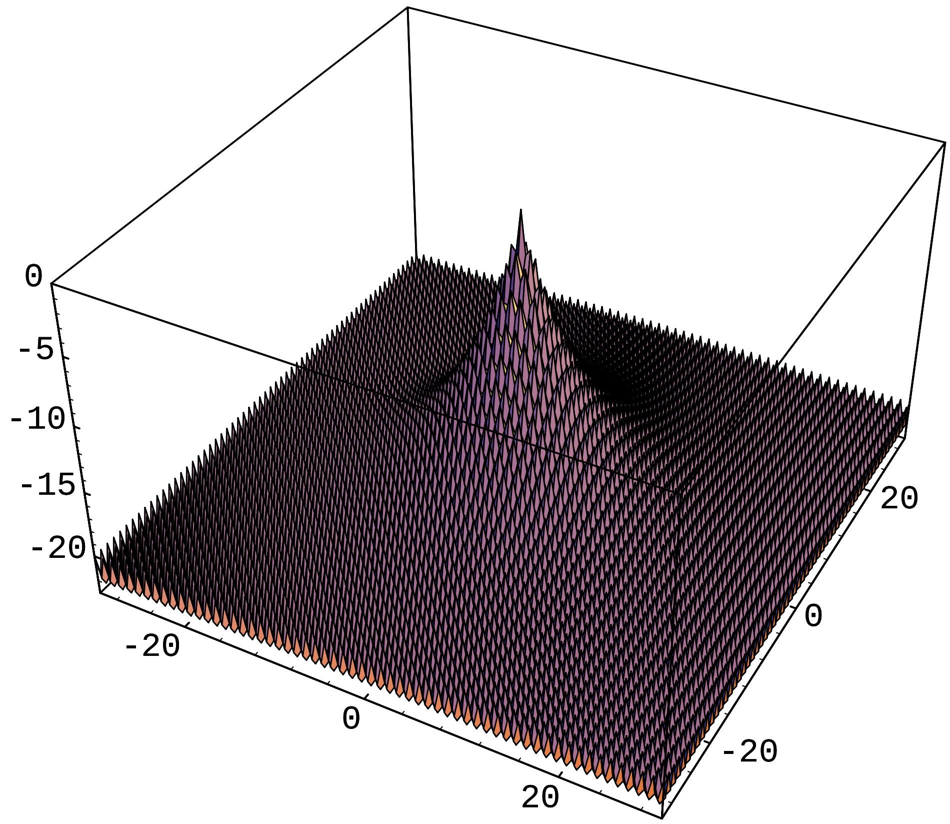


Figure B.1: Inverted Ackley function

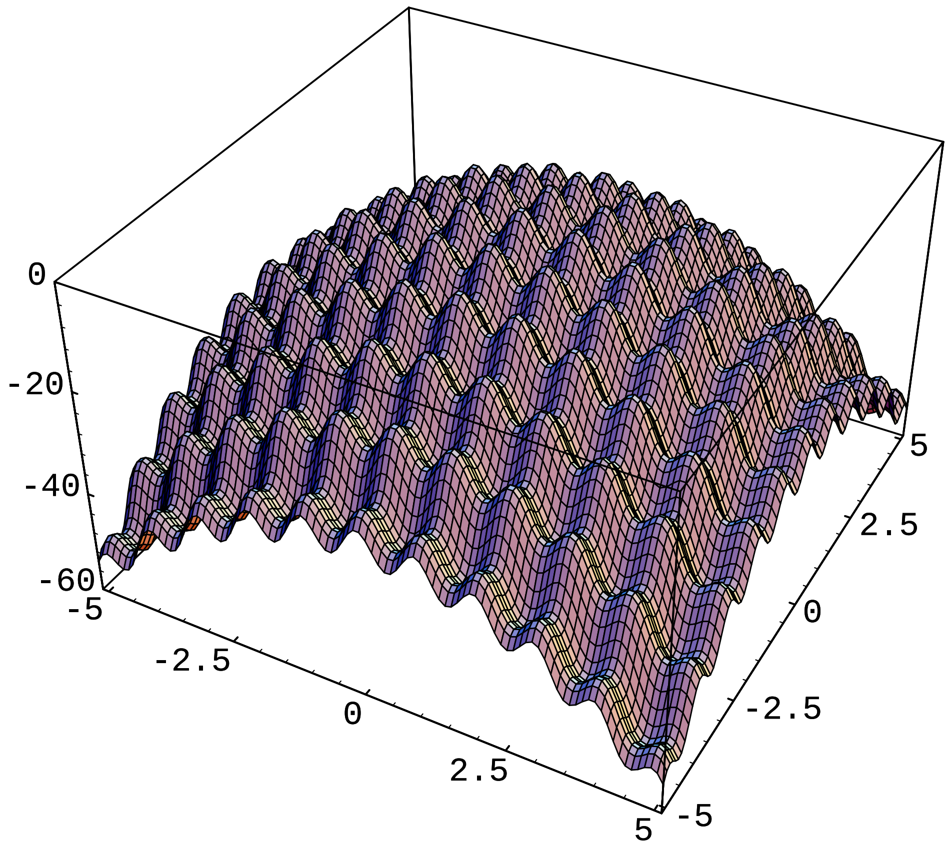


Figure B.2: Inverted Rastrigin function

Schwefel Function

The landscape of the Schwefel function (Schwefel 1981) is covered with a lattice of large peaks and basins. The predominant characteristic of the function is the presence of a second-best minimum far away from the global minimum—intended to trap optimization algorithms on a suboptimal peak. The best minimums are near the corners of the space. We have added the term $418.9829n$ to the Schwefel function so its global minimum will be zero, regardless of dimensionality.

Objective function:

$$f(\vec{x}) = 418.9829n + \sum_{i=1}^{n} x_i \sin\left(\sqrt{|x_i|}\right)$$

Constraints: $-500.0 \leq x_i \leq 500.0$

Minimum: $\vec{x} = (-420.9687, -420.9687, \cdots), \quad f(\vec{x}) = 0.0$

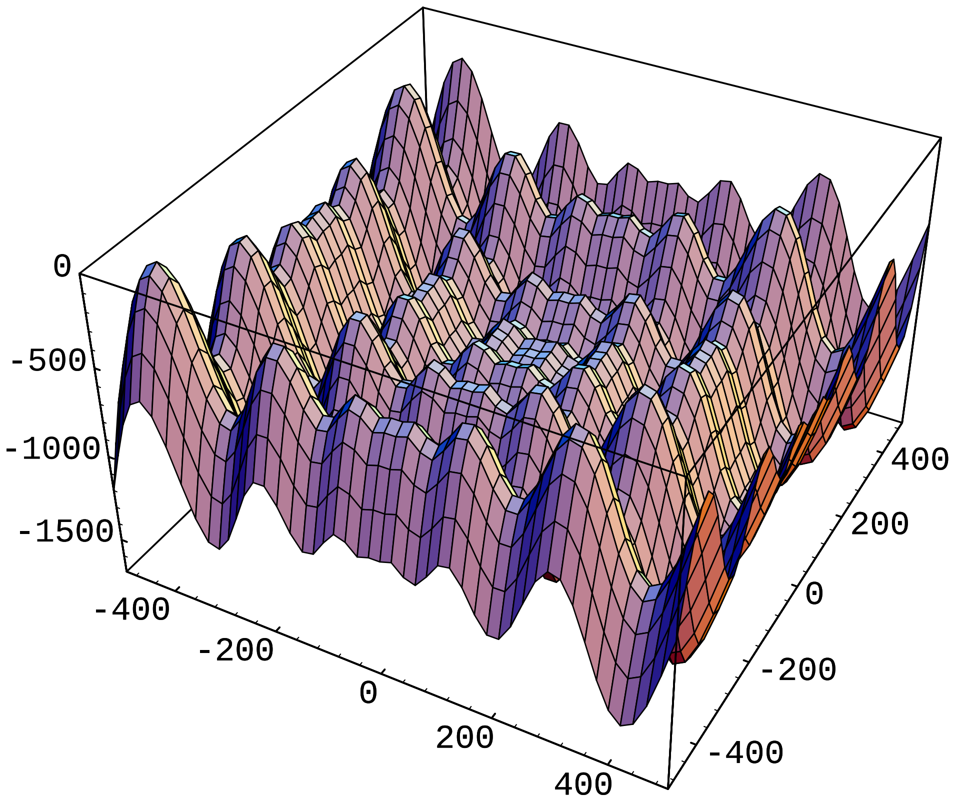


Figure B.3: Inverted Schwefel function

Rosenbrock Function

Rosenbrock (1960) proposed a function of two variables that is characterized by an extremely deep valley whose floor forms a parabola $x_1^2 = x_2$ that leads to the global minimum. Given the nonlinear shape of the valley floor, a simple rotation of the axes does not make the problem significantly easier. The extended version of this function described here was proposed by Spedicato (1975). Similar versions were proposed by Oren (1974) and Dixon (1974).

Objective function:

$$f(\vec{x}) = \sum_{i=1}^{n/2} \left[100(x_{2i} - x_{2i-1}^2)^2 + (1 - x_{2i-1})^2\right]$$

Constraints: $-2.048 \leq x_i \leq 2.048$

Minimum: $\vec{x} = (1, 1, \cdots), \quad f(\vec{x}) = 0.0$

Sphere Model

This function is a very simple quadratic with hyperspherical contours. It has been used previously both in the development of evolution strategy theory (Rechenberg 1973) and in the evaluation of genetic algorithms as part of the De Jong test suite (De Jong 1975).

Objective function:

$$f(\vec{x}) = \sum_{i=1}^{n} x_i^2$$

Constraints: $-5.12 \leq x_i \leq 5.12$

Minimum: $\vec{x} = (0, 0, \cdots), \quad f(\vec{x}) = 0.0$

Stochastic De Jong Function

De Jong (1975) proposed a high-dimensional unimodal quadratic function with Gaussian noise for evaluating the performance of "genetic adaptive plans".

Objective function:

$$f(\vec{x}) = \sum_{i=1}^{n} i x_i^4 + \mathrm{Gauss}(0, \sigma)$$

Constraints: $-1.28 \leq x_i \leq 1.28$

Minimum (without noise): $\vec{x} = (0, 0, \cdots), \quad f(\vec{x}) = 0.0$

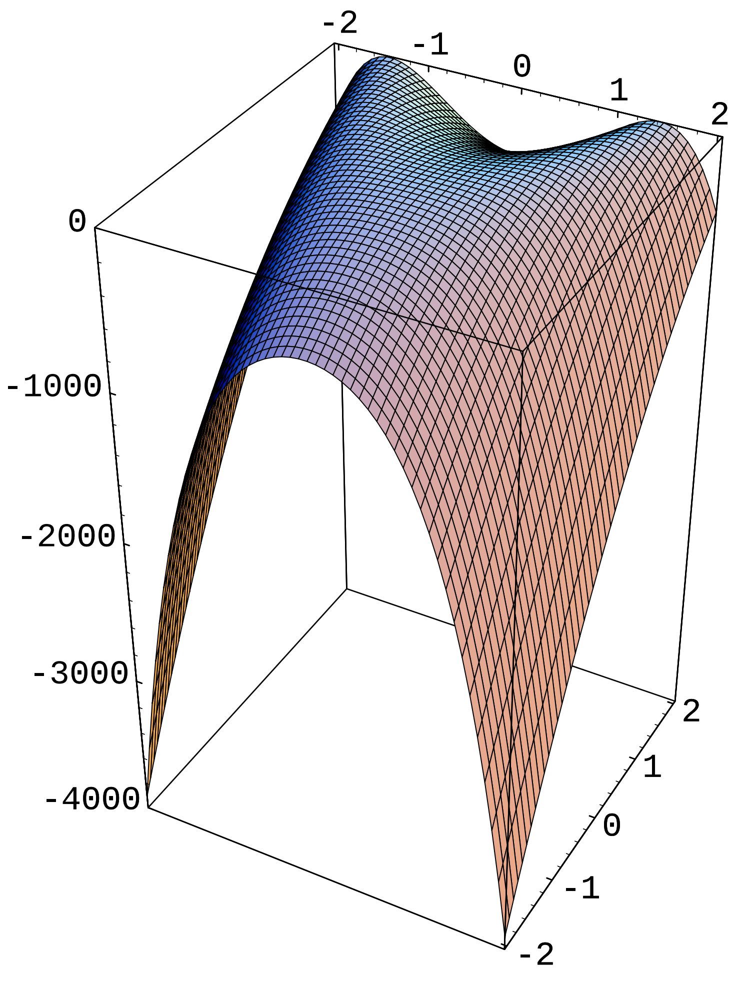


Figure B.4: Inverted Rosenbrock function

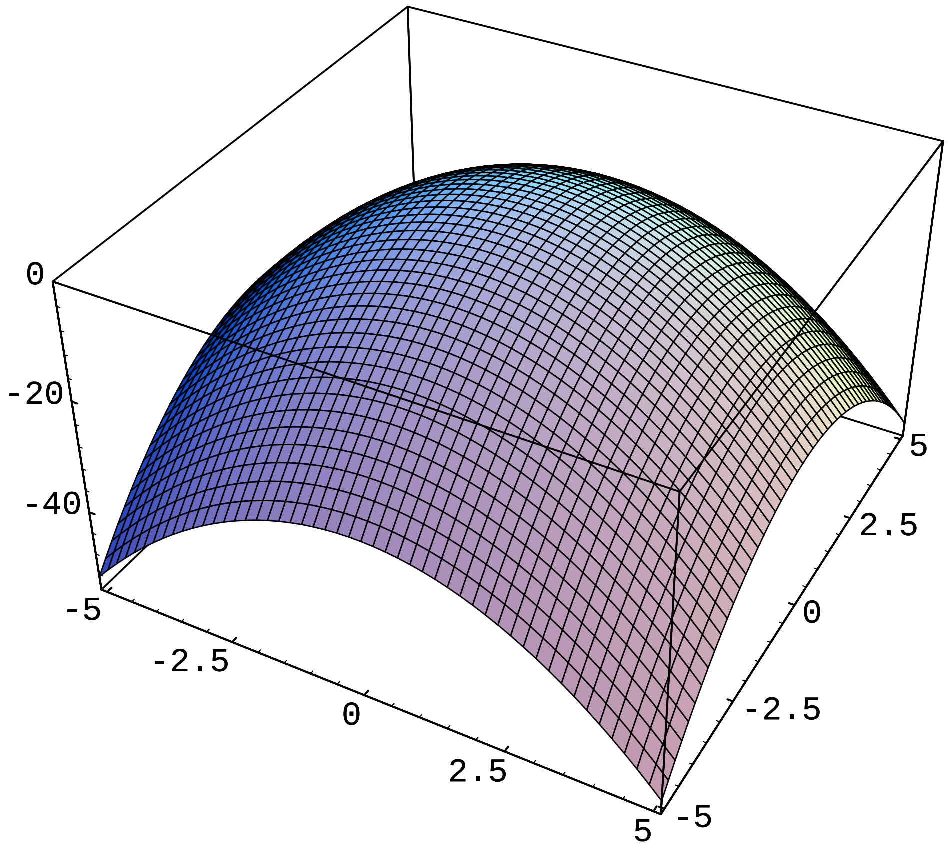


Figure B.5: Inverted sphere model

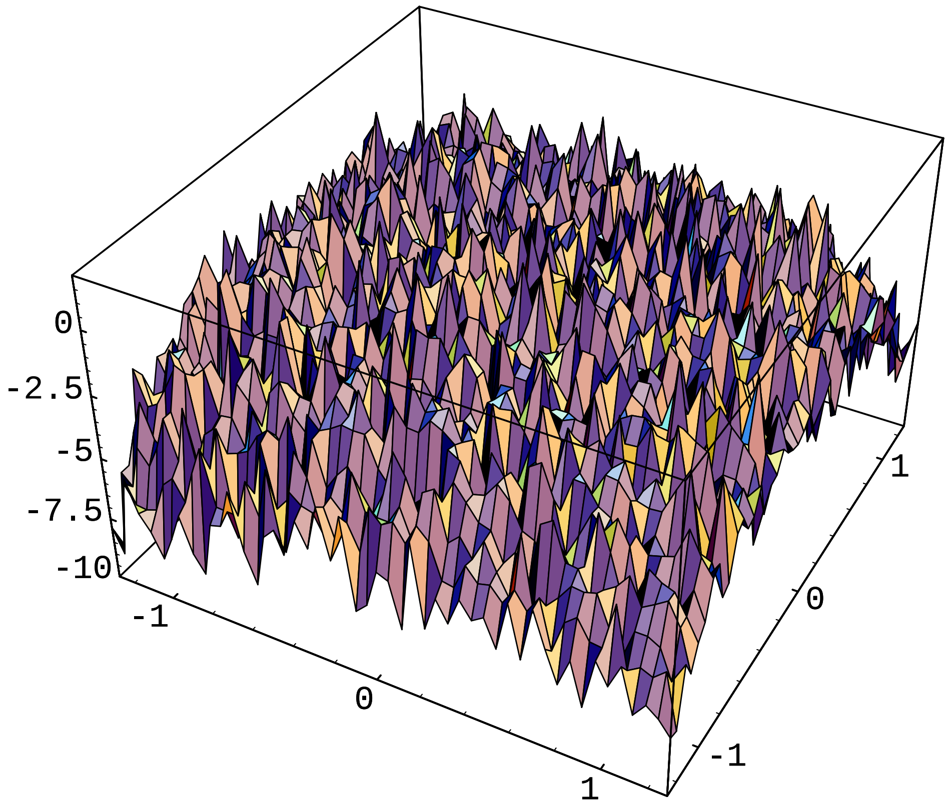


Figure B.6: Inverted stochastic De Jong function ($\sigma = 1.0$)

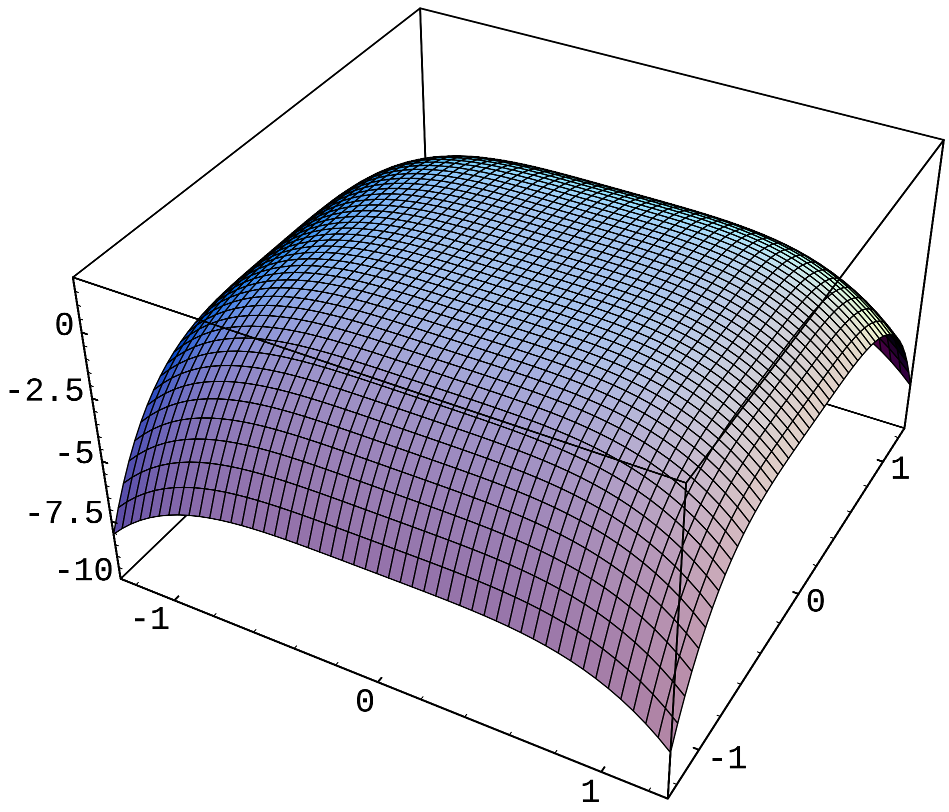


Figure B.7: Inverted stochastic De Jong function with noise removed

Appendix C

PROGRAM CODE FOR COORDINATE ROTATION ALGORITHM

In this appendix, we document Lisp code that implements Salomon's (1996) algorithm for coordinate rotation about multiple axes. This algorithm was used in some of the experiments of chapter 4 to produce massively non-separable functions from separable ones.

Lisp Support for Matrices

The Common Lisp language does not include support for matrix arithmetic (Steele 1990). The following two routines implement the multiplication of square matrices and the creation of identity matrices—both of which are required by the Salomon algorithm. In our implementation, matrices are represented by vectors of vectors, where each sub-vector represents a matrix row.

```
(defun matrix-mult (matrix1 matrix2)
  "Multiply two square matrices of equal dimension"
  (let ((n (array-dimension matrix1 0))
        matrix3
        sum)
    (setq matrix3 (make-array n))
    (dotimes (i n)
      (setf (aref matrix3 i) (make-array n :element-type 'float)))
    (dotimes (i n)
      (dotimes (j n)
        (setq sum 0)
        (dotimes (k n)
          (incf sum (* (aref (aref matrix1 i) k)
                       (aref (aref matrix2 k) j))))
        (setf (aref (aref matrix3 i) j) sum)))
    matrix3))

(defun identity-matrix (n)
  "Return an n-dimensional identity matrix"
  (let ((matrix (make-array n)))
    (dotimes (i n)
      (setf (aref matrix i) (make-array n :element-type 'float)
            (aref (aref matrix i) i) 1.0))
    matrix))
```

Transformation Matrices

The following two routines create transformation matrices for rotating the coordinate system. The angle of rotation is random.

```
(defun transformation (i j n)
  "Return n-dimensional transformation matrix for single rotation"
  (let ((matrix (identity-matrix n))
        (angle (* (- (random 1.0) 0.5) (/ pi 2))))
    (setf (aref (aref matrix i) i) (cos angle)
          (aref (aref matrix j) j) (cos angle)
          (aref (aref matrix i) j) (sin angle)
          (aref (aref matrix j) i) (- (sin angle)))
    matrix))

(defun multiple-transformations (n)
  "Return n-dimensional transformation matrix for multiple rotations"
  (let ((matrix (identity-matrix n)))
    (dotimes (i (- n 1))
      (setq matrix
            (matrix-mult matrix (transformation 0 (1+ i) n))))
    (dotimes (i (- n 2))
      (setq matrix
            (matrix-mult matrix (transformation (1+ i) (1- n) n))))
    matrix))
```

Coordinate System Rotation

The following routine randomly rotates a vector of coordinates about multiple axes. The argument `transformation-matrix` is a rotation matrix previously generated by the routine `multiple-transformations` documented above.

```
(defun rotate (transformation-matrix coordinate-vector)
  "Return vector of rotated coordinates"
  (let* ((n (array-dimension transformation-matrix 0))
         (rotated-coordinates (make-array n :element-type 'float)))
    (dotimes (i n)
      (let ((sum 0.0))
        (dotimes (j n)
          (incf sum (* (aref (aref transformation-matrix i) j)
                       (aref coordinate-vector j))))
        (setf (aref rotated-coordinates i) sum)))
    rotated-coordinates))
```