

An Examination of Coevolutionary Learning

Nathan Lawrence Williams
B.S., Wheaton College (2000)

A thesis presented to the faculty of the
OGI School of Science & Engineering
at Oregon Health & Science University
in partial fulfillment of the
requirements for the degree of
Master of Science
in
Computer Science and Engineering

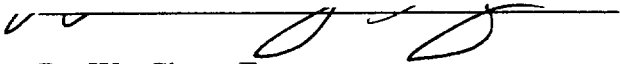
October 2004

Copyright 2004, Nathan Williams

© 2004 Nathan Williams

The thesis "An Examination of Coevolutionary Learning" by Nathan Williams
has been examined and approved by the following Examination Committee:

Dr. Melanie Mitchell
Associate Professor
Thesis Research Advisor



Dr. Wu-Chang Feng
Assistant Professor

Dr. Todd Leen
Professor

Table of Contents

Abstract	viii
1. Introduction	1
2. Problem Description	5
2.1 Evolutionary Computation	5
2.2 Genetic Programming.....	6
2.3 The Problem.....	7
3. Evolutionary Methods and Results	13
3.1 Evolutionary Computation	13
3.2 Previous Results	16
3.3 Results.....	17
4. Analysis	24
4.1 Comparison of Results	24
4.2 Results Analysis	25
5. Conclusion	31
6. Future Work	33
References	34

List of Tables

3.1	Results of the three different evolutionary methods in the original research by Pagie and Hogeweg	16
3.2	Results of the five different evolutionary methods	17
3.3	Results of the size of hosts evolved by the five difference evolutionary methods averaged over 50 runs.....	19

List of Figures

2.1	Example genetic programming representation for an exclusive-or logic gate	6
2.2	Examples of genetic programming crossover and mutation	8
2.3	Plots of the target function	9
2.4	Example host representation for the equation $((X \% Y) + -0.356)$ in the genetic programming paradigm	10
3.1	Quality of all the stable correct hosts produced by the runs of all the fitness evaluation methods	18
3.2	Graph of the true fitness of the best host in each generation for 50 runs using coevolution.....	20
3.3	Graph of the true fitness of the best host in each generation for 50 runs using global evaluation coevolution	20
3.4	Graph of the true fitness of the best host in each generation for 50 runs using random evaluation	21
3.5	Graph of the true fitness of the best host in each generation for 50 runs using resource sharing	21
3.6	Graph of the true fitness of the best host in each generation for 50 runs using complete evaluation.....	22
4.1	Graph of the entropy of the phenotype of the host population at each generation for four example runs of coevolution	25
4.2	Graph of the entropy of the phenotype of the host population at each generation for four example runs using global evaluation coevolution.....	26
4.3	Graph of the entropy of the phenotype of the host population at each generation for four example runs using resource sharing	26
4.4	Graph of the entropy of the phenotype of the host population at each generation for four example runs using random evaluation.....	27
4.5	Graph of the entropy of the phenotype of the host population at each generation for four example runs of complete evaluation.....	27

4.6	Graph of the concentration of coevolving problems in the problem space for a particular successful run of coevolution.....	29
4.7	Graph of the concentration of coevolving problems in the problem space for a particular run global evaluation coevolution.....	30

Abstract

An Examination of Coevolutionary Learning

Nathan Lawrence Williams, B.S.

M.S., OGI School of Science & Engineering

at Oregon Health & Science University

October 2004

Thesis Advisor: Dr. Melanie Mitchell

The goal of this research is to explore *coevolutionary learning*, a variant of genetic algorithms in which candidate solutions co-evolve with the problems on which they are tested. The idea is that the candidate solutions, called “hosts”, improve over time via evolution, while at the same time the problems on which they are tested, called “parasites”, evolve to become increasingly difficult by learning to exploit specific weaknesses in the hosts. Previous research has shown that in a number of cases, coevolutionary learning is more successful than traditional genetic algorithms, but the conditions for such success are not yet understood, nor are the detailed mechanisms underlying the success of coevolutionary learning. Pagie and Hogeweg (Pagie & Hogeweg, 1997) used a simple function approximation problem to compare coevolutionary and non-coevolutionary learning methods in order to study the characteristics of each method. This research replicates and extends the original work in order to explore why coevolution is more successful than traditional genetic algorithms.

This research extends the original research by (1) attempting a replication of the original results, (2) comparing the original evolutionary methods (coevolution, complete evaluation, and random evaluation) with two new methods, global evaluation coevolution and resource sharing, and (3) attempting to understand what causes the different success rates of the different methods, and in particular, why coevolution is so much more successful than any of the other methods.

Chapter 1

Introduction

In biological systems, organisms interact with each other and compete for survival in a changing environment. Through the process of natural selection, the organisms most adapted to their competition and environment survive to breed and spread their genes. Thus, the offspring of the most fit individuals have a combination of the characteristics of both fit parents. Over successive generations the offspring of the individuals are able to continually adapt to their environment.

Inspired by natural selection in biological evolution, evolutionary computation attempts to evolve solutions to problems using *simulated evolution*. With genetic algorithms, a population of solutions is evaluated according to some static fitness measure. The individuals are then selected in proportion to their fitness to proceed to the next generation so that the individuals with higher fitness make up a higher percentage of the next generation's population. The selected individuals are used to create a new population with some new individuals formed via mutation and recombination of the selected individuals. This sequence of evaluation, selection, and recombination is repeated for many generations until either an acceptable solution is discovered or a specified number of generations has passed and no acceptable solution is discovered.

Hillis (1990) was the first to propose the notion of coevolutionary learning. In his scheme, a population of candidate solutions coevolved with a population of test cases. The fitness of an individual in one population depends upon how well it solves, in one

case, or stumps, in the other case, individuals in the other population. The particular model Hillis explored coevolved sorting networks and sorting problems. This adaptation of standard simulated evolution is often referred to as “host-parasite” or “predator-prey” coevolution because the fitness of one population is based upon how well it defeats the other population. In Hillis’ system, the fitness of an individual sorting network depended upon how many sorting problems in an individual group it successfully sorted. Likewise, the fitness of an individual sorting-problem group depended upon how many problems in the group a network did not successfully sort. Thus, there is a complementary relationship between the hosts and the parasites. Hillis embedded the populations of sorting problems and sorting networks on a spatial lattice so that the populations were only evaluated on the individuals in the same grid cell. This coevolution scheme was able to discover a smaller correct sorting network than standard evolution.

More recently, Mitchell and Pagie in their work, “A Comparison of Evolutionary and Coevolutionary Search” (2002), studied evolving cellular automata on the density classification task. In the density classification task, a learner attempts to discover a set of cellular automata rules that can correctly classify bit strings according to density (if the bit string has a majority of 1’s, then the bit string belongs to density class 1 and vice versa). They evolved the cellular automata rules on a spatial lattice. Their results showed that 86% of their runs using coevolution discovered high fitness strategies, while only 2% of their runs using a standard evolutionary approach evolved high fitness strategies.

The studies above, among others, have shown the benefits of coevolution to overcome some of the major issues in machine learning. In theory, coevolution provides an adaptable gradient for learning so that the populations evolve to continually provide a challenge for one another. Additionally, it is thought that the populations are able to focus on the strategic weaknesses in the other population so that each population learns to generalize its solution to solve the complete set of problems rather than just a specific sub-set of problems upon which it is evaluated. This means that coevolution has the potential to produce optimal trainers for different stages of learning. Also, since coevolution does not provide a static fitness measurement, a problem theoretically has no

upper limit on its fitness as the populations push each other towards continual improvement (Watson & Pollack, 2001). Finally, as shown in this work, coevolution allows for sparse training rather than relying on large training sets. This is because the training set adapts so that each learner does not need to be evaluated on a training set that covers the full range of possible situations a learner encounters. Thus, coevolution can provide computational efficiency of learning.

However, some studies have shown that coevolution does not lead to better results for every task. In some problem domains, coevolution can lead to a loss of gradient. This happens when one coevolving population evolves to greatly outperform the other population so that the fitness of the other population provides no comparative fitness information (i.e., all individuals in the population completely fail so that the evolutionary procedure cannot determine which individuals performed better than the others). Additionally, for some tasks coevolution can lead to over-fitting, in which one population focuses on the weaknesses in the other population without generalizing the solution. Finally, coevolution can lead to relativism, in which the populations oscillate between strategies because each population's fitness is defined by the other population's fitness, and we therefore have no absolute measure of fitness (Watson & Pollack, 2001). This relativism is exhibited when one population can solve problem A but not problem B. Thus, the other population focuses on problem B. The first population must then evolve to solve problem B; however, if the population forgets how to solve problem A in the process, then the second population will focus on problem A. This oscillating behavior does not result in true progress towards learning the full problem domain.

Previous research has shown that in a number of cases, coevolutionary learning is more successful than traditional genetic algorithms, but the conditions for such success are not yet understood, nor are the detailed mechanisms underlying the success of coevolutionary learning. Here we report results from an extension to research on coevolution originally done by Pagie and Hogeweg. In both the original and the present study, the most striking difference between traditional evolutionary search and coevolution is their success rate. For example, in our study the coevolutionary model succeeded in producing acceptable

solutions in 78% of the simulation runs, while the standard evolutionary model was unable to produce any acceptable solutions. We also compared these results with those of other techniques in an attempt to determine what aspects of coevolutionary search underlie its success. Random evaluation is used to determine the effect sparse evaluation has on producing successful solutions. Resource sharing is used to examine the role of population diversity. Global evaluation coevolution is used to indicate the role adaptive problem gradient plays in successfully evolving correct solutions.

Our hypotheses are that coevolution promotes continued diversity in the population and produces parasites that target specific weaknesses in hosts. Continued diversity would create a greater probability that a crossover or mutation produces a new host with a higher fitness. Parasites that target specific weaknesses in hosts would cause the hosts to adapt to solve those problems in order to survive. We believe that these are the two factors that produce the success of coevolution.

Chapter 2

Problem Description

2.1 Evolutionary Computation

Fitness is the driving force of Darwinian natural selection and, likewise, of genetic algorithms. John Holland conceived genetic algorithms (Gas) and gave a theoretical framework for adaptation in his 1975 book *Adaptation in Natural and Artificial Systems*.

In GAs, candidate solutions to a particular problem are encoded as a string of features appropriate to the problem domain. This string of features serves as the candidate solution's "genetic code". This genetic code is what is used to calculate a candidate solution's fitness. A GA uses a single population of these candidate solutions which is randomly generated to start. As a result, each individual in the initial population typically has very low fitness. However, some of the individuals in the population will have slightly better fitness than others, and the genetic representation of these individuals will be chosen for the next generation so that they will become the basis for further advances toward solving the problem.

Each generation the whole population of candidate solutions is evaluated for fitness. The most fit individuals are then chosen as the basis for the next generation's population. Some of the chosen individuals are combined together using crossover, which takes the genetic code of two individuals and combines them to create a new individual, and some of the chosen individuals are modified by mutation, which randomly changes the value of a single feature to another random value. This process of evaluation, selection, and

modification repeats until a solution is discovered that meets predetermined performance criterion and the run succeeds, or a specified number of generations have passed and no solution has been discovered and the run fails.

2.2 Genetic Programming

Traditional evolutionary computation has been performed using populations represented as linear lists of features often represented as bit strings (usually constant length, but sometimes variable). However, for some domains this representation is not conducive to representing solutions whose final size, shape, and complexity are largely unknown.

Therefore, Koza introduced the genetic programming paradigm in his work, "Genetic Programming: A Paradigm for Genetically Breeding Populations of Computer Programs to Solve Problems" (1990). With this

model, solutions are hierarchically structured computer programs represented as trees. Each node in the tree is drawn from a set of functions, each of which takes a specific number of arguments (functions that have no arguments are considered a special class called terminals). Depending upon the problem, the functions may be standard arithmetic operations (such as addition, subtraction, multiplication, and division), or control functions (such as If-Then-Else and Do-While), or even domain specific functions (such as Temperature or

Turn-Left). The domain of representation, therefore, is the set of all possible trees that can be composed using the functions specified for the problem domain. Note that this representation is similar to how computers internally represent programs as they compile them from human computer languages to machine readable programs. Figure 2.1 shows an example genetic programming representation of the Boolean exclusive-or operation.

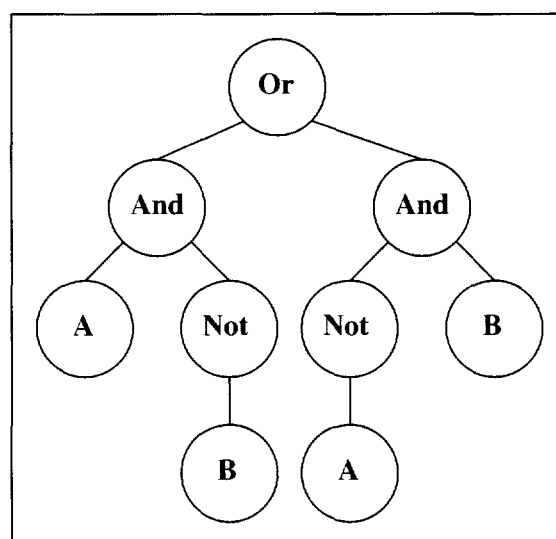


Figure 2.1: Example genetic programming representation for an exclusive-or logic gate using the function set { Or, And, Not } and the terminal set { A, B }.

Populations of genetic programming individuals are initialized as random trees. A tree is generated by first choosing a random function as the root node for the new tree. Then, for every argument the function requires, a new random function is chosen as the root of the sub-tree rooted at the new node. This process is repeated for each sub-tree. This recursive initialization of genetic programming trees is often restricted to a specific maximum height (i.e. the node randomly chosen for a node at the maximum height from the root of the tree must be a terminal node).

There are two genetic programming operations used by the evolutionary process to evolve trees: mutation and crossover. Examples of both of these operations are shown in Figure 2.2.

Mutation:

Mutation is an operation designed to primarily continue to add diversity to the population. There are various types of mutation that have been employed for genetic programming. The type utilized in this work is called point mutation which modifies a single tree. Under point mutation, a node is randomly chosen from the tree and is replaced by another random node that takes the same number of arguments. That is, a function that takes two arguments can only be mutated into another random function that also takes two arguments. Likewise, a terminal can only be mutated into another randomly chosen terminal.

Crossover:

Crossover is the method through which portions of trees are recombined in order to potentially move highly fit sub-trees together and thus create even fitter trees. The operation of crossover involves two trees. A random node is selected in the first and in the second tree. A copy of the sub-tree rooted at the chosen node in the second tree then replaces the sub-tree rooted at the chosen node in the first tree.

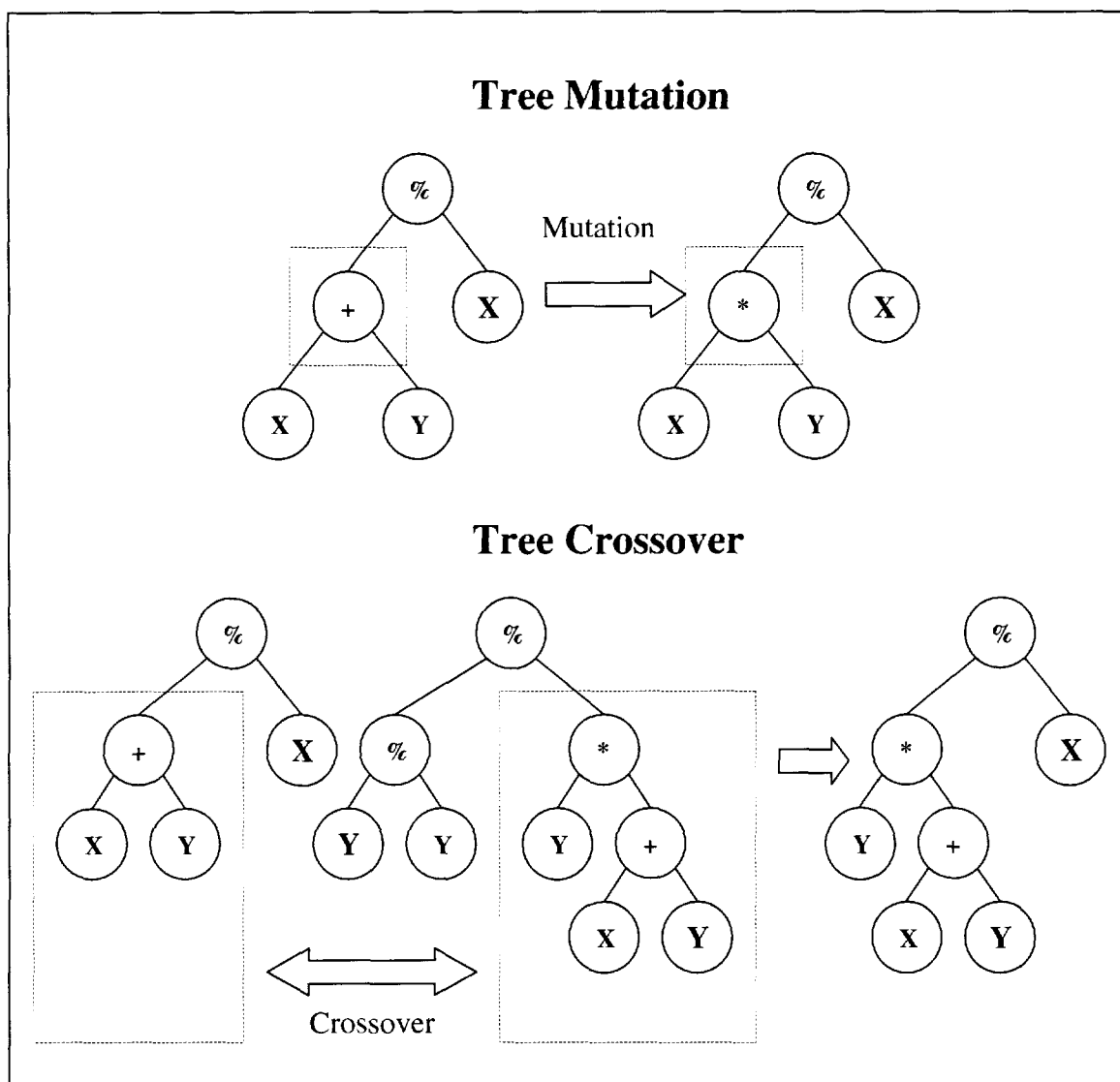


Figure 2.2: Examples of genetic programming crossover and mutation. In Tree Mutation, the protected division function node (%) was randomly chosen and mutated into the multiplication function (*). In Tree Crossover, the sub-tree rooted at the protected division function was a randomly chosen along with the sub-tree rooted at the multiplication function. A copy of the first tree is made with the tree selected in the first tree replaced by a copy of the sub-tree selected in the second tree.

2.3 The Problem

The task presented to the evolutionary or coevolutionary system is to evolve mathematical equations. The target function is the following two-dimensional numerical equation: $f(x,y) = 1/(1+x^{-4}) + 1/(1+y^{-4})$. A plot of the target function is shown in Figure 2.3. Each potential solution (or “host”) is a mathematical equation whose goal is to approximate the target equation within a certain tolerance. The problems upon which fitness is based are simple (x, y) values evenly distributed over the problem domain:

$x \in [-5.0, 5.0]$, $y \in [-5.0, 5.0]$ at regular intervals of 0.4. Therefore, the set of complete problems is a 26 x 26 problem grid and contains 676 total problems.

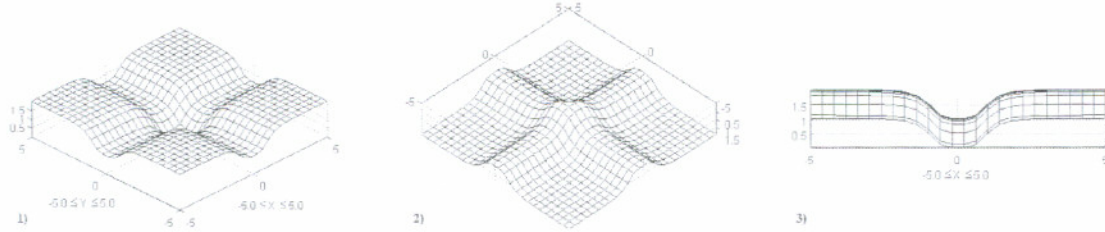


Figure 2.3: Plots of the target function with views from 1) the top, 2) the bottom, and 3) the side. The maximum of the function approaches 2.0 at the corners and the minimum approaches 0 at (0, 0).

The *fitness* of a host is based upon its weighted error when evaluated on a subset of problems. The particular subset of problems upon which the hosts are evaluated depends upon the evolutionary method used. The *true fitness* of a host is the average error of the host on the complete set of 676 problems.

True Fitness: $tf_h = \left(\sum_{i=1}^{S_{\text{Parasites}}} |t(p_i) - h(p_i)| \right) / S_{\text{Parasites}}$, where $S_{\text{Parasites}}$ is the number of problems in the complete set of problems, $t(p_i)$ is the value of the target function on problem p_i , and $h(p_i)$ is the value of a host on problem p_i .

The hosts are represented, using the genetic programming paradigm, as trees describing mathematical equations that can be evaluated on two-dimensional points as illustrated in Figure 2. The function set is $\{+, -, *, \%$ and the terminal set is $\{x, y, \mathfrak{R}\}$. The $+$, $-$, and $*$ operators are the standard arithmetic functions for addition, subtraction, and multiplication respectively; each takes two arguments. The protected divide function ($\%$) (Koza, 1992) is a function of two arguments, A and B, and returns 1 if $B = 0$ and A / B otherwise. The terminals X and Y evaluate to the respective components of the coordinates on which the parse tree is being evaluated. The terminal \mathfrak{R} is the “ephemeral random constant” (Koza, 1992) which is a randomly generated value between -1.0 and 1.0 . The ephemeral random constant’s value for the node is randomly generated when

the node is created and is fixed for the lifetime of the node. Note that the function and terminal set contains unnecessary elements. In fact, the only necessary elements are the terminals X and Y, protected division, and either addition or subtraction. Additionally, while the constant 1.0 is not directly available as a terminal, it can be created by dividing a number by itself or by dividing a number by zero. The initial population of hosts is randomly generated with a maximum depth of three nodes.

Figure 2.4 provides an example of a representation of an equation using the genetic programming paradigm. The instance of the ephemeral random constant (\mathfrak{R}) was randomly set as -0.356 when the node was created, so that the represented equation is $((X \% Y) + -0.356)$.

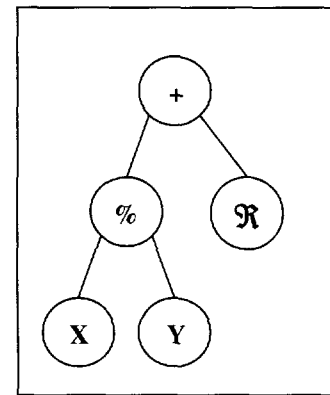


Figure 2.4: Example host representation for the equation $((X \% Y) + -0.356)$ in the genetic programming paradigm. The constant -0.356 is the value of this particular instance of the ephemeral random constant (\mathfrak{R}).

The coevolutionary methods utilize a population of “parasites” describing problems. The parasites are simply (x, y) coordinate pairs from the complete set of all problems. These parasites are used to define problems on which hosts are evaluated. The fitness of a parasite is defined by the evolutionary method used. The initial members of the parasite population are randomly chosen with replacement from the complete set of 676 possible problems. Note that Pagie and Hogeweg initialized all parasites to (0.2, 0.2). However, we obtained better results for the coevolutionary methods when the parasites were randomly initialized from the complete set of possible problems.

Contrary to standard evolutionary computation which keeps all candidate solutions in a single population, the host and the parasite populations are spatially embedded on a 50 x 50 two-dimensional toroidal lattice, with one host and one parasite per grid cell. Thus, the total population sizes are 2500 individuals. At each generation, the host population undergoes selection, crossover, and mutation, and the parasite population undergoes selection and mutation. These are described below.

Selection:

Competition between individuals for survival in a population is local in space. Thus, each host or parasite competes for survival against the surrounding 8 hosts or parasites in its 3 x 3 neighborhood. After each population has been evaluated to determine fitness using one of the evolutionary methods, the individuals in each 3 x 3 neighborhood are ranked according to their fitness values. The i th ranked individual is selected with probability $\left(\frac{1}{2}\right)^i$. The ninth element is selected with probability $\left(\frac{1}{2}\right)^8$ to ensure that the nine probabilities sum to unity. The selected individual replaces the individual in the center of the neighborhood.

Crossover:

Following the selection step, 40% of the selected host population is chosen with replacement to undergo crossover with a randomly chosen neighboring host (including itself) from among the individuals it competed with for selection. As illustrated in Figure 3, host crossover replaces a randomly chosen sub-tree in one host with a copy of a randomly chosen sub-tree from another host, leaving the first host modified but the second host unchanged. There is no parasite crossover.

Mutation:

Following the crossover step, 20% of the selected host population is chosen with replacement to undergo point mutation. As illustrated in Figure 3, host point mutation replaces a random tree node with another randomly chosen tree node that takes the same number of arguments. Thus, a terminal node is replaced by another randomly chosen terminal node, or a function node is replaced by another randomly chosen function node. Additionally, at each generation 10% of the parasites are chosen to undergo mutation. Parasite mutation modifies either the x or y component of the coordinate pair by adding or subtracting one step (0.4). However, when a parasite on the edge of the problem space is mutated so that its new value would be outside of the problem space, the parasite remains unchanged.

A host is considered to be *correct* if for each of the 676 problems in the problem set, the absolute difference between the target function and the host function is less than or equal to a tolerance of 0.01 ($\forall p, p \in \text{Parasites}, |t(p) - h(p)| \leq 0.01$). A *stable* correct host is one that has been discovered and then remains in the population for at least 50 generations. The algorithm stops when a stable correct host has been discovered or after 500 generations if none are discovered by that time. The goal of each run is to produce a stable correct host; therefore, runs that do so are considered *successful*.

The hosts have a rich representation space which allows them to represent equivalent equations with different trees. The particular way a host represents its equation is called the genotype. The actual set of problems the host solves is its phenotype. Thus the hosts can have many genotype encodings for a single phenotype. In this work, we define phenotype as the set of problems for which the error of the host with respect to the target function is less than or equal to the tolerance of 0.01.

Host Phenotype: the set of all p such that $p \in \text{Parasites}$ and $|t(p) - h(p)| \leq 0.01$

Chapter 3

Evolutionary Methods and Results

This research compares a variety of evolutionary methods. We replicate the original methods used by Pagie and Hogeweg (1997): coevolution, complete evaluation, and random evaluation. However, we also add two more methods, global evaluation coevolution and resource sharing, for comparison. We chose resource sharing to examine the role of diversity in the population.

We report on 50 runs for each method. In order to compare the various methods, at each generation we store the host with the best true fitness, the coevolving problems (if any), and the number of hosts in each generation that solve each problem.

3.1 Evolutionary Methods

Below we define fitness functions of hosts and parasites for each of the different evolutionary methods that we investigate. Note that since a host's fitness is defined in terms of its error with respect to the target function, each evolutionary method attempts to minimize host fitness f_h . In addition, each coevolutionary method attempts to maximize parasite fitness f_p , which is defined as the error of local host h when evaluated on p .

Coevolution: Let $p = (x, y)$, $x \in [-5.0, 5.0]$, $y \in [-5.0, 5.0]$, be a particular problem.

Host fitness: $f_h = \left(\sum_{i=1}^9 |t(p_i) - h(p_i)| \right) / 9$

The fitness of a host is the host's average error on the nine problems defined by the neighboring parasites, where $t(p)$ is the target function value on problem p and $h(p)$ is the host function's value on problem p . The fitness evaluation is considered *sparse* because it does not evaluate a host against the complete set of problems, but rather just a subset.

$$\text{Parasite fitness: } f_p(h) = |t(p) - h(p)|$$

The fitness of a parasite is calculated with respect to the host h in the same grid cell and is the absolute value of the error of the host on that problem. Note that while a host's fitness is a function of its value on the nine parasites in its neighborhood, a parasite's fitness depends only on the single host in its same grid location. Pagie and Hogeweg claim that such an asymmetric fitness evaluation of parasites produces better results, in terms of time to discover a good solution, than a symmetric fitness evaluation in which the parasite's fitness would depend upon the nine neighboring hosts (Pagie & Hogeweg, 1997).

Complete Evaluation: Complete evaluation is the standard evolutionary method to which coevolution is compared. Under complete evaluation there is no parasite population. Rather, we attempt to minimize the host's mean error when the host is evaluated against all 676 problems in the problem space (S_{Problems}) with equal weighting.

$$\text{Host fitness: } f_h = \left(\sum_{i=1}^{S_{\text{Problems}}} |t(p_i) - h(p_i)| \right) / S_{\text{Problems}}$$

Random Evaluation: Random evaluation was chosen as a control experiment for coevolution to determine if coevolution's success as compared with complete evaluation was due only to sparse evaluation (evaluation against only a subset of the problem space) rather than other aspects of coevolution. Under random evaluation there is no parasite population, but sparse evaluation is still maintained. For each generation and for each

host, nine random problems are chosen with replacement from the complete problem set on which each host's fitness is calculated.

$$\text{Host fitness: } f_h = \left(\sum_{i=1}^9 |t(p_i) - h(p_i)| \right) / 9$$

Global Evaluation Coevolution: Global evaluation coevolution was chosen as a control experiment to determine whether coevolution's success comes from its ability to provide an adaptive gradient for the whole population rather than using spatial locality to dynamically exploit specific weaknesses in specific hosts. Under global evaluation coevolution, a parasite population exists, but parasites are no longer local to their hosts. Instead, each host is evaluated on all 2500 parasites in the parasite population ($n_{\text{parasites}}$). However, as in coevolution, the parasites still attempt to maximize the error of the host in the same grid cell; therefore, some locality is maintained.

$$\text{Host fitness: } f_h = \left(\sum_{i=1}^{n_{\text{parasites}}} |t(p_i) - h(p_i)| \right) / n_{\text{parasites}}$$

$$\text{Parasite fitness: } f_p(h) = |t(p) - h(p)|$$

Resource Sharing: Resource sharing attempts to promote diversity in the population by giving higher weight to problems that fewer hosts solve compared with problems that many hosts solve (Rosin & Belew, 1995; Juille & Pollack, 1998). Thus, resource sharing was chosen as a control experiment to determine the role population diversity plays in evolving stable correct solutions. Resource sharing provides an alternative problem-weighting mechanism theorized to behave similarly to how parasites exploit the hosts during coevolution. The credit each host earns for solving a particular problem is shared equally among all the hosts that solve that particular problem. Thus, the more hosts that get a problem correct, the less credit that each host receives for solving that problem.

$$\text{Host fitness: } f_h = \sum_{i=1}^{S_{\text{Problems}}} \left(\text{weight}(p_i) * \text{covered}(h, p_i) \right), \text{ where}$$

$$\text{weight}(p) = 1 / \left(\sum_{i=1}^{n_{\text{hosts}}} \text{covered}(h_i, p) \right), \text{ and}$$

$$\text{covered}(h, p) = \begin{cases} 1 & \text{if } |t(p) - h(p)| \leq 0.01 \\ 0 & \text{if } |t(p) - h(p)| > 0.01 \end{cases}$$

The fitness of a host is the sum of the weights of each problem in the complete set of problems on which the host matches the target function within the accepted tolerance. The weight(p) of each problem p depends upon how many other hosts solve p : the fitness a host receives from solving the problem is shared among all hosts that solve it. The function $\text{covered}(h, p)$ returns 1 if the error of host h on problem p is less than the accepted tolerance of 0.01, and 0 otherwise.

3.2 Previous Results

The original results by Pagie and Hogeweg are summarized in Table 3.1. Pagie and Hogeweg were able to evolve stable correct hosts approximately 45% of the time using coevolution. However, complete evaluation did not produce any stable correct hosts. Their results indicate that sparse evaluation can result in successfully evolving the complete solution to an evolutionary target. Random evaluation produced a success rate of 35%, which is roughly comparable to the success rate of standard coevolution. Pagie and Hogeweg reasoned that in both of these methods, sparse evaluation allows the evolutionary process freedom to explore the solution space more freely than does complete evaluation because it forces the hosts to focus on reducing the error on a subset of problems while allowing it to ignore others until later rather than forcing the hosts to optimize the error on all problems at once. Therefore, they hypothesized that the freedom afforded by sparse evaluation was the variable that led to the success of evolving correct hosts using the random and coevolving methods.

Table 3.1: Results of the three different evolutionary methods in the original research by Pagie and Hogeweg. The success rate is the percentage of runs out of 20 total that produced a stable correct host.

Evolutionary Method	Success Rate	Mean Number of Nodes in Final Host
Coevolution	9 / 20 (45%)	44
Complete Evaluation	0 / 20 (0%)	68
Random Evaluation	7 / 20 (35%)	not reported

3.3 Results

We performed 50 runs using each of the five methods described above. These 50 runs provide a more statistically significant success rate than the 20 runs previously done. The results are summarized in Table 3.2. Each run was independent and was generated using a different random number seed. The success rate of a method is the number of runs that produced a stable correct host out of the total number of runs. During each simulation, we recorded the best individual in that generation based upon true fitness in order to provide a basis of comparison between the different evolutionary methods. Ties between hosts for the best individual in the population based upon true fitness were broken by which host had been stable the longest, then by which host had the fewest number of nodes, and finally by which host was encountered first. The best individual in the population was chosen from all the individuals tied for the best true fitness by first choosing the individuals in this group that had been stable the longest, and from these choosing the individuals with the fewest number of nodes, and finally from these choosing the first individual encountered in a pass over the spatial grid.

Table 3.2: Results of the five different evolutionary methods. The success rate is the percentage of runs out of 50 total that produced a stable correct host. The High Quality Hosts are the number of stable correct hosts evolved with a true fitness less than 10^{-8} .

Evolutionary Method	Success Rate	High Quality Hosts
Coevolution	39/50 (78%)	25/39
Complete Evaluation	0/50 (0%)	0/0
Random Evaluation	7/50 (14%)	5/7
Global Evaluation Coevolution	26/50 (52%)	12/26
Resource Sharing	6/50 (12%)	1/6

Basic coevolution outperformed all other methods significantly with a success rate of 78%. The next best method proved to be global evaluation coevolution with a 52% success rate. Resource sharing and random evaluation proved to perform approximately equally well with a 12% success rate for resource sharing and a 14% success rate for random evaluation. Complete evaluation did not produce any stable correct hosts. Therefore, it appears that coevolution methods are able to achieve a much higher success

rate than the other methods (random evaluation, resource sharing, and complete evaluation).

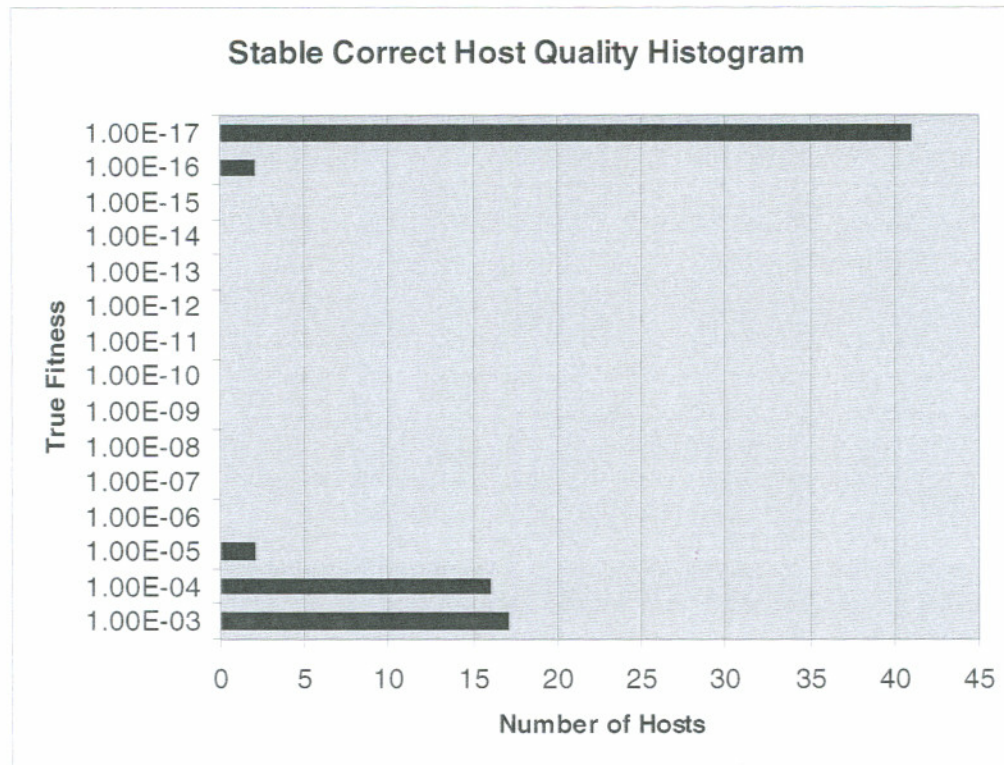


Figure 3.1: Quality of all the stable correct hosts produced by the runs of all the fitness evaluation methods.

Additionally, there were two clusters of successful host types found with a natural break between them as shown in Figure 3.1. The first group was composed of hosts that had a true fitness between 10^{-2} and 10^{-5} , while the second, called high quality hosts, had true fitness less than 10^{-15} . These high quality hosts indicated that the run had produced an exact solution versus an approximate solution. Using coevolution, half of all runs produced high quality hosts, while global evaluation coevolution produced high quality hosts in only 24% of the runs and in less than half of the successful runs.

Table 3.3: Results of the size of hosts evolved by the five different evolutionary methods averaged over 50 runs. The mean number of nodes in stable correct hosts is the mean number of nodes in all runs that produced stable correct hosts.

Evolutionary Method	Mean Number of Nodes in Final Hosts	Standard Deviation	Mean Number of Nodes in Stable Correct Hosts	Standard Deviation
Coevolution	101.28	133.56	60.83	53.77
Complete Evaluation	1301.92	1320.22	N/A	N/A
Random Evaluation	1548.76	2242.02	72.43	52.28
Global Evaluation Coevolution	628.58	824.65	166.81	255.06
Resource Sharing	995.44	1597.92	125.33	120.53

The results of the size of the evolved hosts are summarized in Table 3.3. The size of the hosts in a population directly affects run time due to the time needed to calculate the fitness of every host in the population. Thus, methods that produce smaller hosts run faster and consume fewer resources. Coevolution produced significantly smaller hosts as compared with the other methods. However, the mean number of nodes in the final program is somewhat misleading because it largely depends upon how many stable correct hosts are found in 50 runs, since the number of nodes in the stable correct hosts is generally less than the number of nodes in the final incorrect hosts. However, all methods have an extremely wide range of host sizes, as indicated by the standard deviations. Nevertheless, coevolution produced the smallest stable correct hosts.

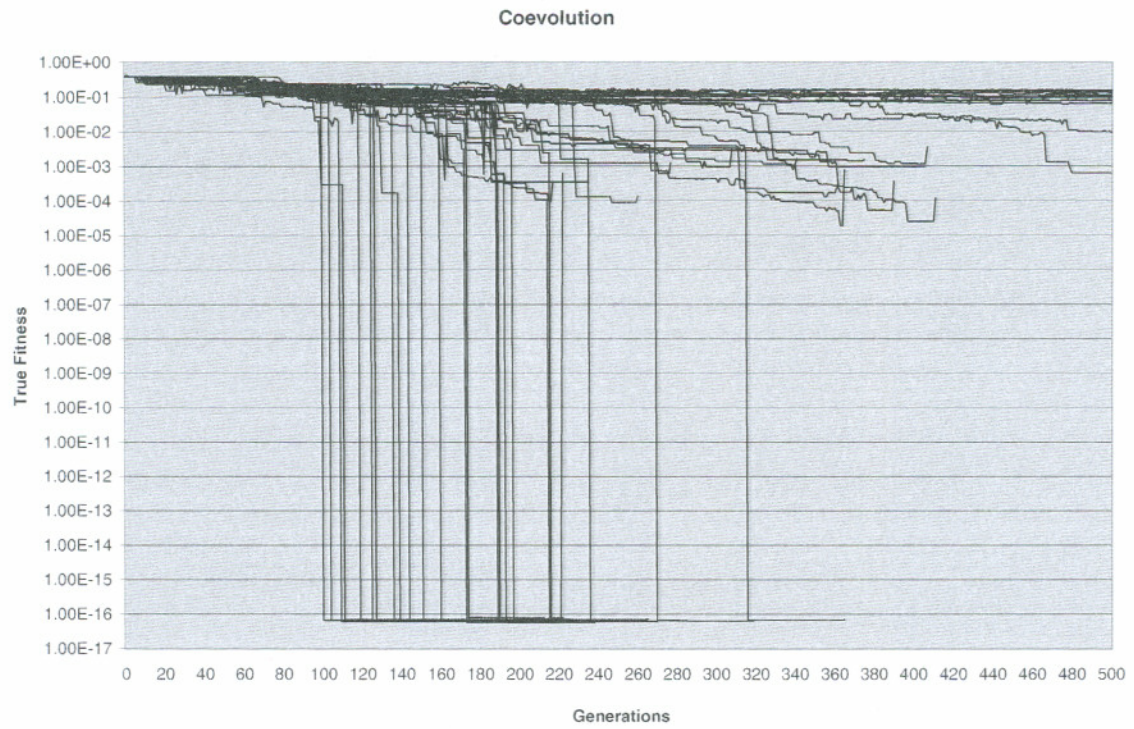


Figure 3.2: Graph of the true fitness of the best host in each generation for 50 runs using coevolution.

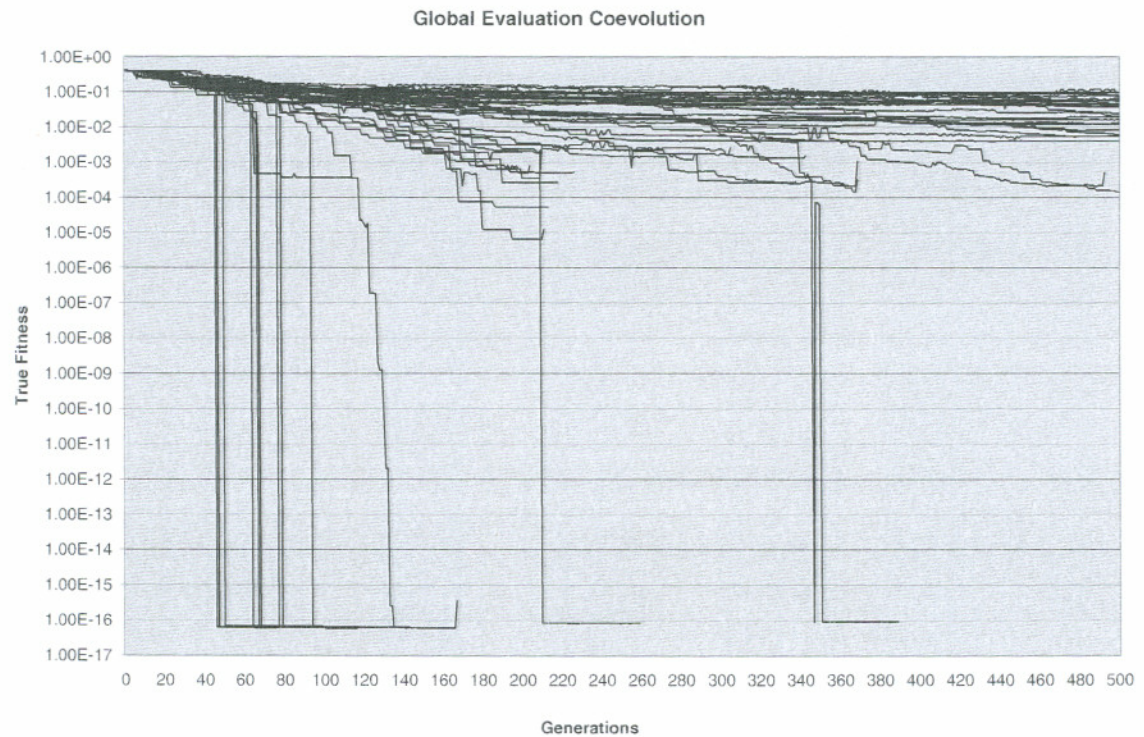


Figure 3.3: Graph of the true fitness of the best host in each generation for 50 runs using global evaluation coevolution.

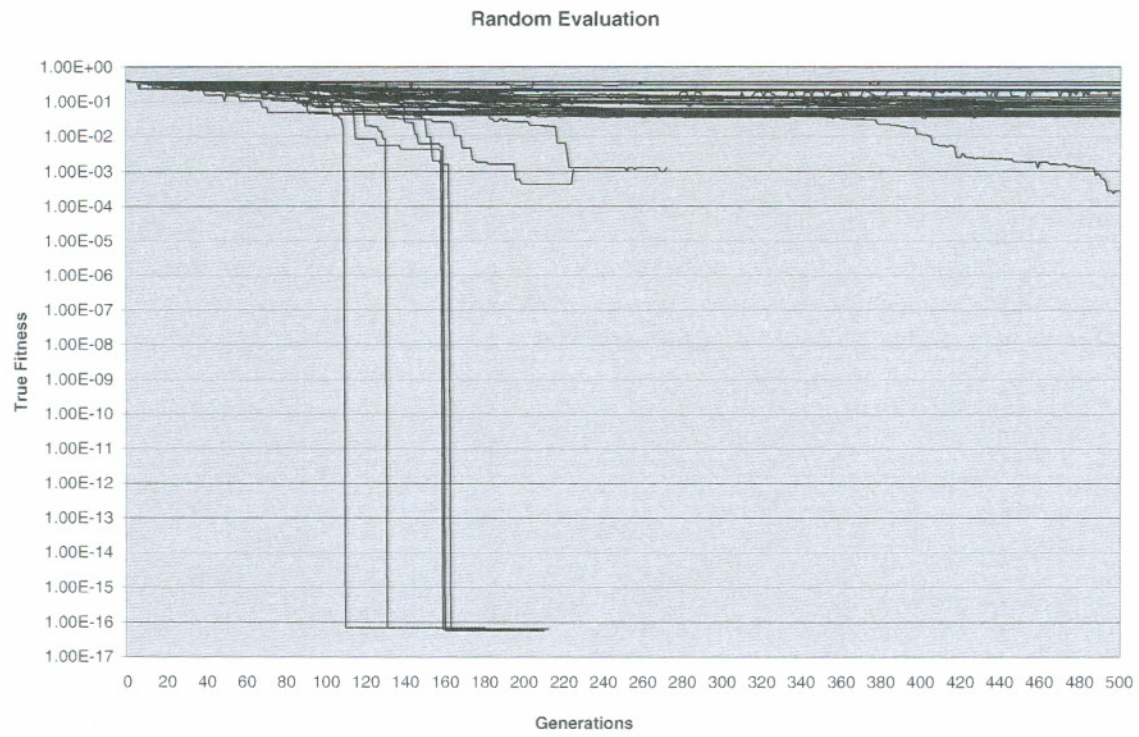


Figure 3.4: Graph of the true fitness of the best host in each generation for 50 runs using random evaluation.

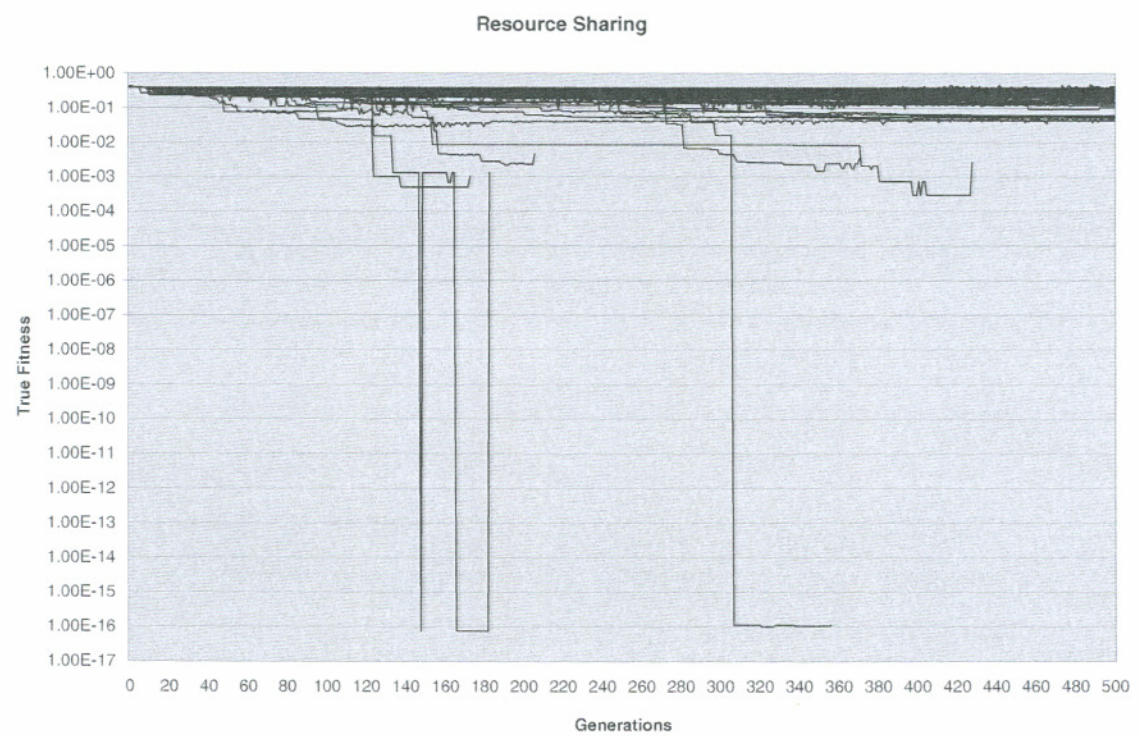


Figure 3.5: Graph of the true fitness of the best host in each generation for 50 runs using resource sharing.

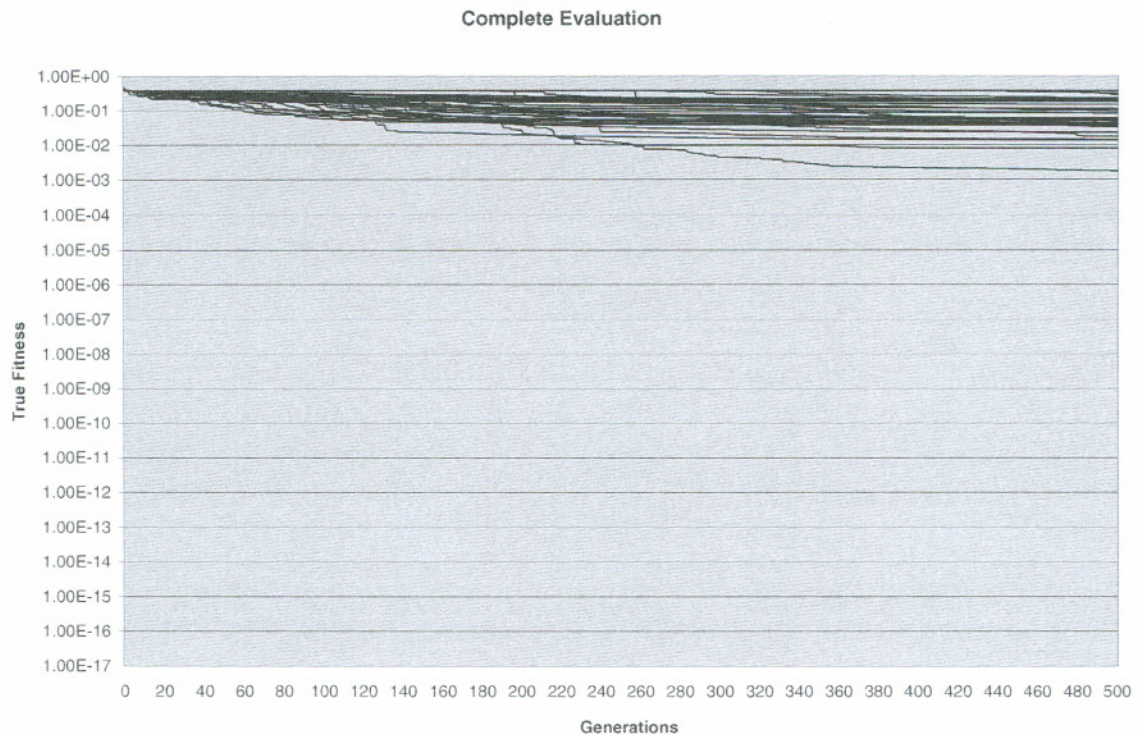


Figure 3.6: Graph of the true fitness of the best host in each generation for 50 runs using complete evaluation.

As shown in Figures 3.2 through 3.6, all evolutionary methods are able to improve the true fitness of the hosts to some degree during each simulation. However, for the most part, the high quality hosts are not gradually evolved, but rather they come from large fitness jumps from lower quality correct or incorrect hosts. In contrast, lower quality hosts are gradually evolved during the simulation and are the result of many small fitness jumps rather than a single large one. In all fitness evaluation methods except complete evaluation, there are times when the fitness evaluation method will cause a small loss of true fitness. That loss is generally temporary, and a host with equal or greater true fitness is subsequently evolved. The larger fitness retrograde jumps at the end of a run are due to a lower quality correct host becoming stable before the host with the lowest true fitness becomes stable, and thus the former becomes the final stable correct host.

Complete evaluation however is notable because it was unable to evolve a single correct solution, as shown in Figure 3.6. While complete evaluation has some improvement in true fitness as the population evolves, there are no large jumps in true fitness of the best

individual in the population as the population evolves; rather, the evolutionary process is only able to do small optimizations to existing individuals.

Chapter 4

Analysis

4.1 Comparison of Results

There are three major differences between the results previously reported by Pagie and Hogeweg (1997): the success rates of coevolution and random evaluation, and the mean number of nodes in the final host.

Pagie and Hogeweg report a success rate for coevolution of 45%, compared with our success rate of 78%. The difference is due to the how the parasites are initialized: Pagie and Hogeweg initialized their parasites to a single central value, and we initialized ours to randomly chosen problems, which we discovered led to better results. When we ran coevolution with parasites Pagie and Hogeweg's initial values, we obtained success rates comparable to theirs.

Pagie and Hogeweg report a success rate for random evolution of 35%, whereas we find that random evolution succeeds in only 14% of runs. We also find that the trees that are evolved in all methods are typically much larger than the sizes reported by Pagie and Hogeweg (see Tables 2 and 3 above). In spite of a detailed review of Pagie and Hogeweg's algorithm, with the assistance of Ludo Pagie (personal communication), we have been unable to identify the cause of these differences.

4.2 Result Analysis

Following Pagie and Mitchell (2002), we investigate the following two hypotheses for why coevolution has such a relatively high success rate as compare with the other methods investigated here:

- 1) Coevolution allows for continued diversity in the population.
- 2) Coevolution produces parasites that target specific weaknesses in hosts.

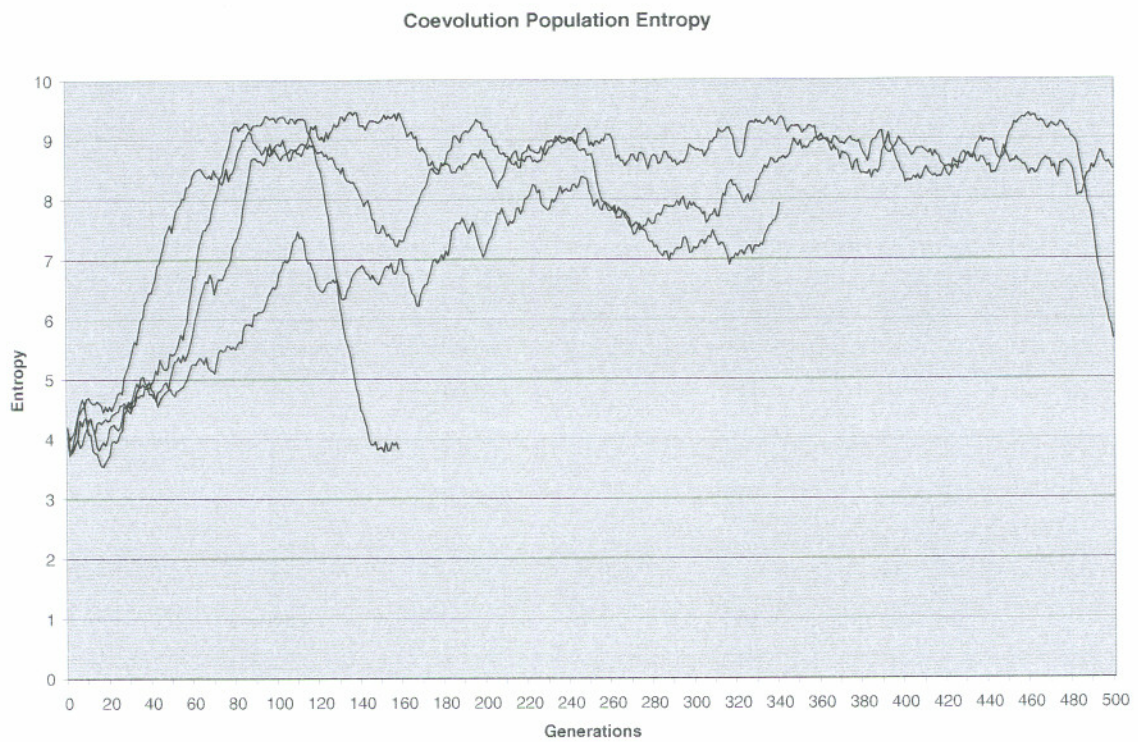


Figure 4.1: Graph of the entropy of the phenotype of the host population at each generation for four example runs of coevolution.

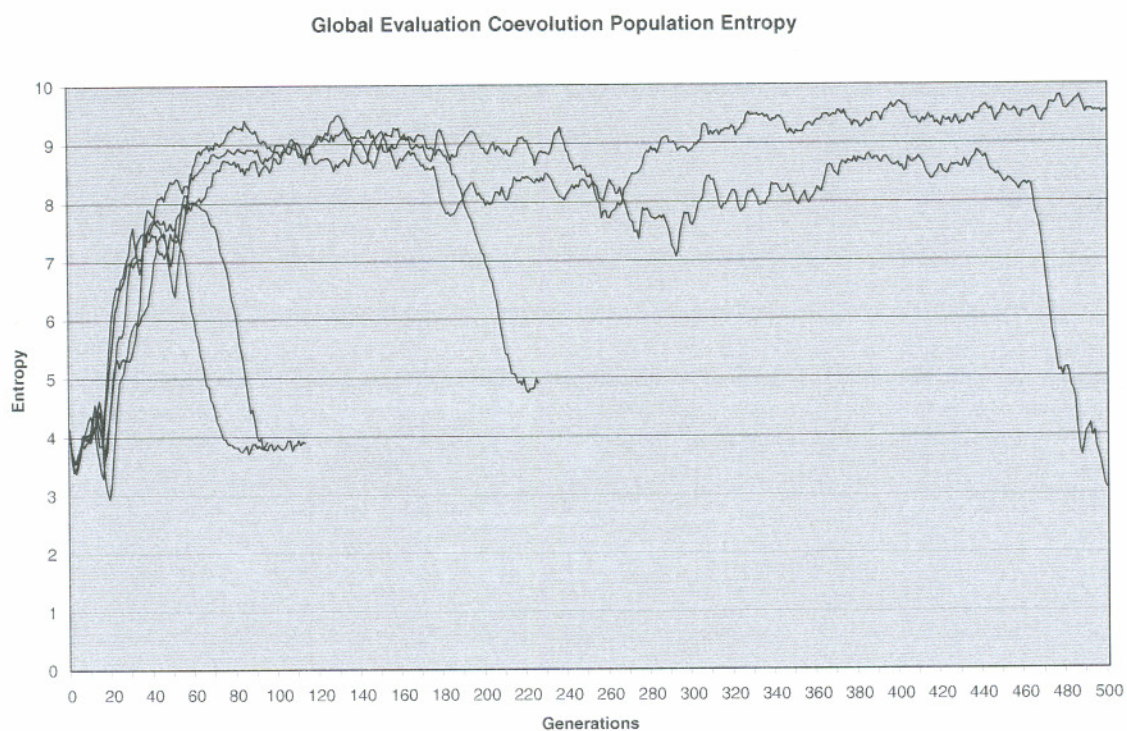


Figure 4.2: Graph of the entropy of the phenotype of the host population at each generation for four example runs of global evaluation coevolution.

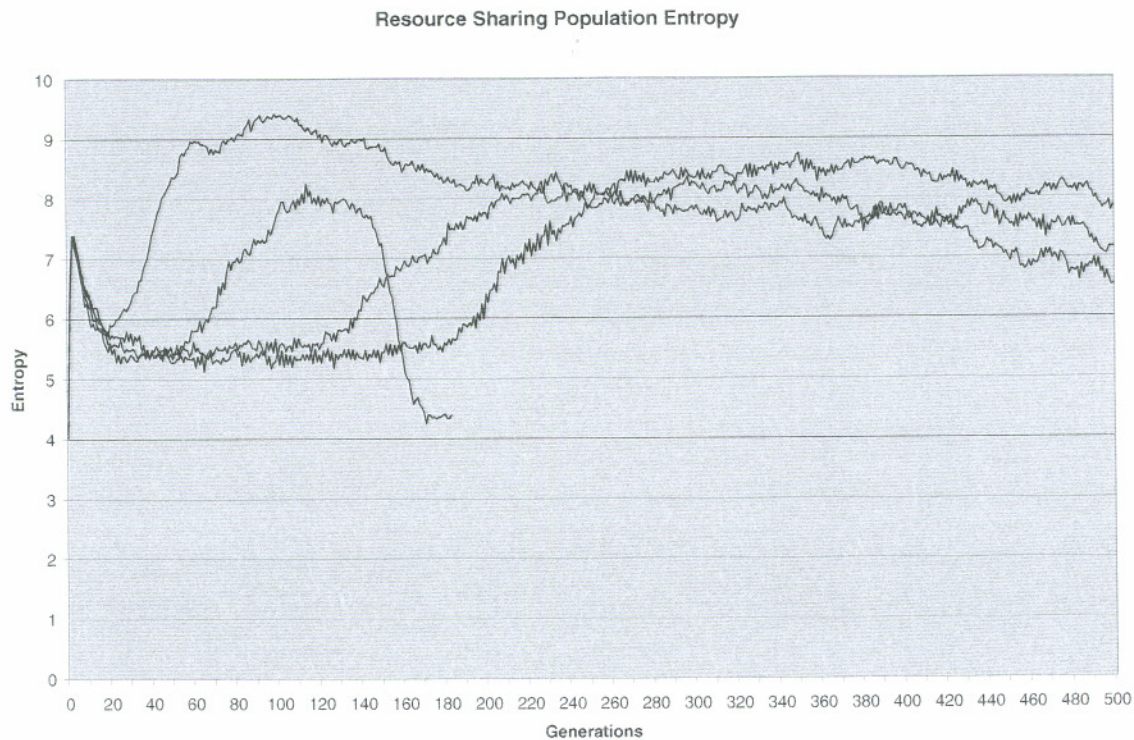


Figure 4.3: Graph of the entropy of the phenotype of the host population at each generation for four example runs of resource sharing.

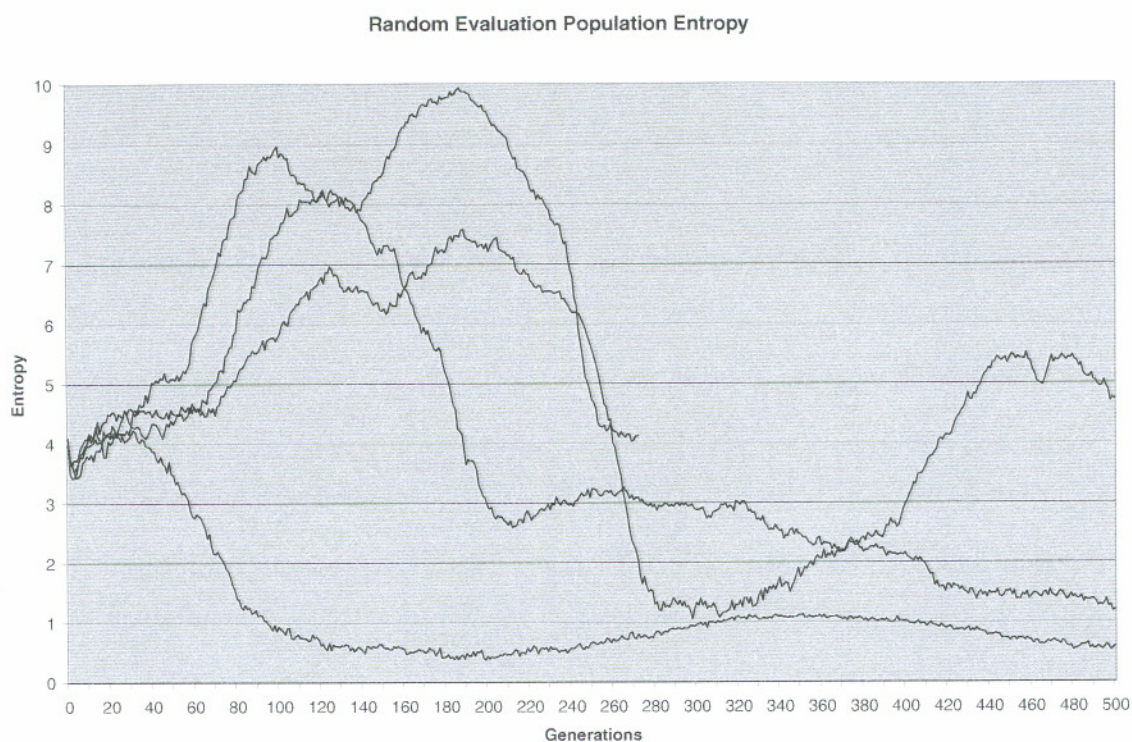


Figure 4.4: Graph of the entropy of the phenotype of the host population at each generation for four example runs of random evaluation.

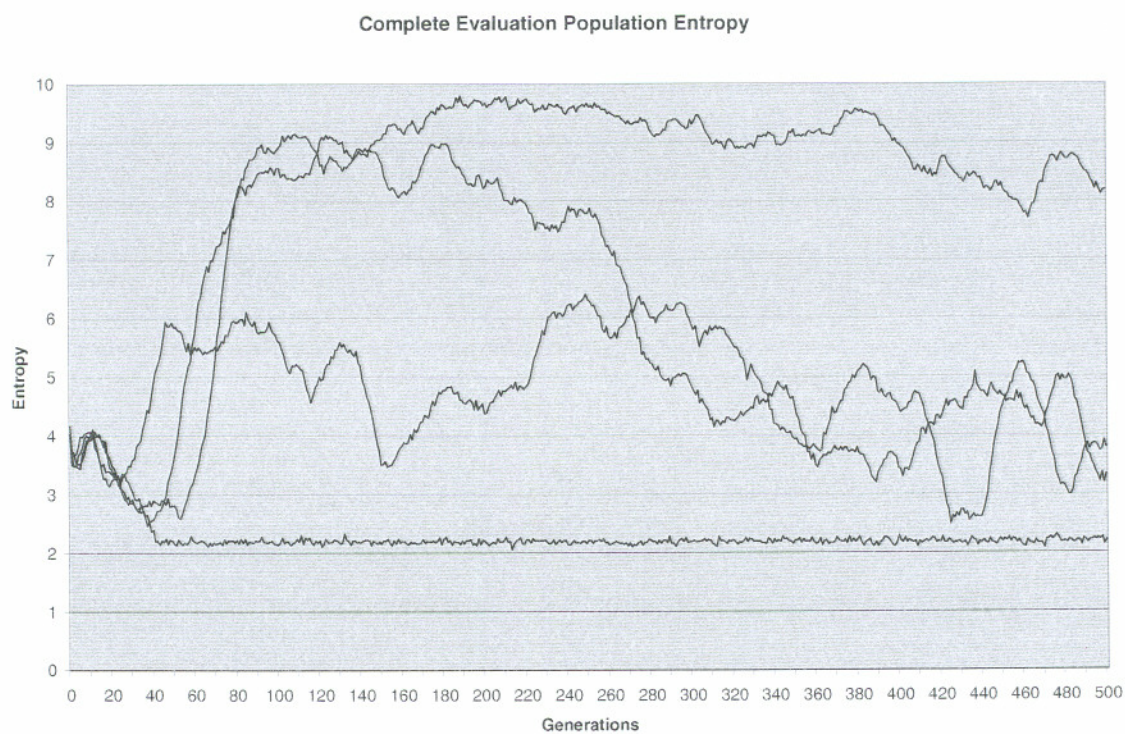


Figure 4.5: Graph of the entropy of the phenotype of the host population at each generation for four example runs of complete evaluation.

Our first hypothesis is that coevolution encourages continued population diversity and this diversity leads to coevolution's success. Diversity is thought to be important because low diversity decreases the chances a useful crossover will occur. This research uses entropy of the different phenotypes exhibited in the population as a measure of diversity. Entropy is a measure of the uniformity of the population's phenotypes; a higher entropy value indicates greater phenotype diversity.

$$\text{Entropy} = \sum_{i=0}^{n_{\text{phenotypes}}} p_i \log_2 p_i, \text{ where}$$

$$p_i = \text{phenotype}_i / n_{\text{hosts}}, \text{ and}$$

phenotype_i is the number of hosts that exhibit a specific phenotype, n_{hosts} is the total number of hosts which is 2500, and $n_{\text{phenotypes}}$ is the number of different phenotypes exhibited in the population.

Figures 4.1 through 4.5 show the phenotype entropy of four different example runs for each of the fitness evaluation methods. Because the host population is randomly initialized, all runs for all evaluation methods start with entropy of approximately 4.0. Coevolutionary methods and global evaluation coevolution both have high phenotype entropy immediately, and the entropy remains high until a correct solution is discovered, at which time the entropy drops as the correct host dominates the population. At times, one strategy will gain momentum to overtake the population. However, the parasites quickly respond and restore the balance by targeting the weakness in the phenotype group. Only when a correct host is discovered does the host phenotype entropy drop as that host takes over the population. Therefore, it appears that one of the strengths of coevolution is that it provides an effective adaptive fitness method that helps maintain population diversity. Resource sharing also encourages high entropy as shown in Figure 4.3, although the entropy does not increase as immediately as the coevolutionary methods, nor does it go quite as high. Additionally, the resource sharing entropy curves have small sharp spikes when compared to the coevolutionary methods, possibly because the resource sharing runs reach stability within the population between the different phenotype groups present in the population. Complete evaluation (shown in Figure 4.4)

has significantly lower entropy values for a majority of the runs. While some of the example runs have an initial gain in entropy, the entropy drops off with successive generations. Because of the high probability that the best individual in each neighborhood will be selected, the best individuals quickly dominate the population, thereby reducing the diversity. Likewise, random evaluation (shown in Figure 4.5) has an initial gain in entropy to levels competitive with the highly successful coevolutionary methods; however, the entropy drops off with successive generations to those comparable with complete evaluation. Therefore, while high sustained phenotype diversity is exhibited by the more successful strategies, a high diversity is not necessarily indicative of highly successful fitness evaluation method.

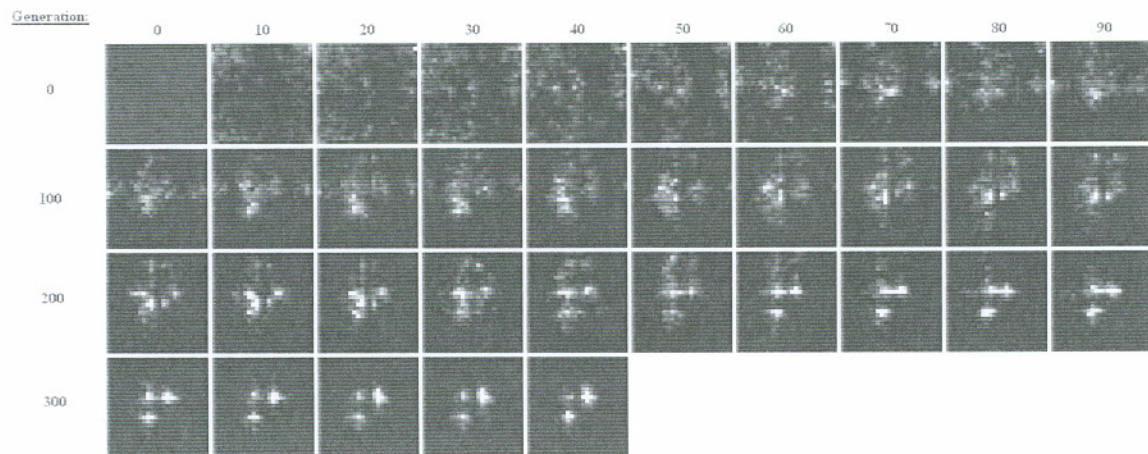


Figure 4.6: Graph of the concentration of coevolving problems in the problem space for a particular successful run of coevolution. Snapshot of parasites taken every 10 generations.

Our second hypothesis is coevolution produces parasites that target specific weaknesses in hosts. Figure 4.6 shows snapshots of the concentration of problems that the parasite population uses to challenge the hosts for a particular successful run. The problems start out scattered over the problem domain and evolve to an early focus on the edges of the problem domain. However, they soon start focusing on specific problems spread throughout the domain, with specific focus towards the center and in the corners. The hosts find it easier to solve the problems in the corners first before they successfully solve the ridges of the target function in the center of the problem domain. Eventually, as a successful host is discovered, all problems focus on the center of the problem domain.

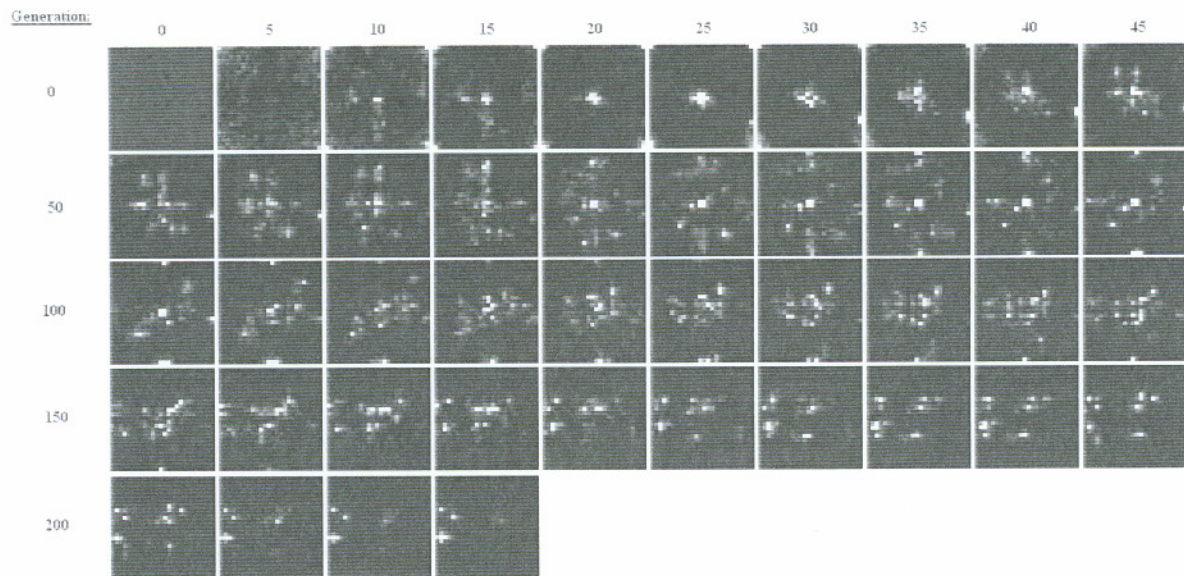


Figure 4.7: Graph of the concentration of coevolving problems in the problem space for a particular run of global evaluation coevolution. Snapshot of parasites taken every 5 generations. A stable correct host was discovered at generation 169.

The parasites in global evaluation coevolution, like in coevolution, also focus on particularly hard problems exhibited by Figure 4.7. The parasites in global evaluation coevolution do not directly affect hosts, rather, they affect the whole population of parasites indicate a set of problems that the whole host population has difficulty solving. Global evaluation coevolution focuses parasites primarily in the center of the problem domain; however, it repeatedly sends out small “particles” of problem groups from the center towards the edges of the problem domain. This indicates relativism in the population caused by hosts that forget how to solve specific problems, and thus the parasites continually revisit those problems. This relativism is most likely caused by the inability of the parasites to target specific weaknesses in local hosts and is likely the reason for global evaluation coevolution’s lower success rate than coevolution.

Chapter 5

Conclusion

Coevolution had a better success rate than all other methods. The sparse fitness evaluation alone was able to produce a little success as shown by the random evaluation method. Likewise, the diversity in a population led to a little success as shown by the resource sharing method. However, the coevolutionary methods used adaptive fitness to target weaknesses in the host population, which not only helped maintain population diversity by focusing on the weaknesses of large phenotype groups when they threatened to dominate the population, but also forced the hosts to adapt. Hosts were able to completely solve the evolutionary target by induction from a few strategic hard problems.

Therefore, it appears that while phenotype diversity is present in the highly successful coevolutionary methods, it does not necessarily lead to high success as shown by resource sharing. However, the first hypothesis that coevolution causes high population diversity is correct. Our second hypothesis that coevolution produces parasites that target specific weaknesses in hosts is also correct. The ability of the parasites to provide an adaptive problem weighting for the different stages of learning the target equation seems to be not only the cause of the phenotype diversity but also a big part of the key to coevolution's success as shown by global evaluation coevolution.

Another factor in the success of coevolution is the locality provided by embedding the host and parasite populations on a spatial lattice so that host fitness is based upon the parasites in a host's immediate neighborhood. This gave coevolution an advantage over global evaluation coevolution since the populations were able to target specific

weaknesses in the local hosts, which appears to reduce the relativism exhibited by the populations while boosting the success rate.

Chapter 6

Future Work

Future work also includes applying these techniques to real-world problems. Coevolution would be especially useful for cases when the problem domain is extremely large but problems can be generated with ease. Thus, an application in regression or problem solving would be an ideal domain of application.

In the future, more work needs to be done further analyzing the properties of the evolving populations. Further diversity measurements and analysis of the growth and spread of emerging strategies will lead to a better understanding of the principles at work.

Additional comparative studies of other fitness methods, such as boosting algorithms, can also lead to a better understanding of the problem domain as it did in this study.

References

- Hillis, D. W. (1990); Co-evolving parasites improve simulated evolution in an optimization procedure. *Physica D* 42:228-234.
- Juillé, H. and Pollack, J. B. (1998). Coevolutionary learning: A case study. *Proceedings of the Fifteenth International Conference on Machine Learning*, Madison, Wisconsin, July 24 - 26, 1998, pp 251-259.
- Koza, J. R. (1990): *Genetic Programming: A Paradigm for Genetically Breeding Populations of Computer Programs to Solve Problems*. Technical Report No. STAN-CS-90-314, Computer Sciences Department, Stanford University.
- Koza, J. R. (1992). *Genetic Programming: On the programming of computers by means of natural selection*. Cambridge, MA: MIT Press.
- Pagie, L. & Hogeweg, P. (1997); Evolutionary consequences of coevolving targets. *Evolutionary Computation* 5(4):401-418.
- Pagie, L. & Mitchell, M. (2002). A comparison of evolutionary and coevolutionary search. *International Journal of Computational Intelligence and Applications*, 2(1), 53--69.
- Rosin, C. D. & Belew, R. K. (1995). Methods for competitive coevolution: Finding opponents worth beating. In Eshelman, L. J. (Ed.), *Proceedings of the Sixth International Conference on Genetic Algorithms*, pp373-381. San Mateo, CA: Morgan Kaufmann.
- Watson, R.A. and Pollack, J.B. (2001). Coevolutionary Dynamics in a Minimal Substrate. *Proceedings of the 2001 Genetic and Evolutionary Computation Conference*, Spector, L, et al (eds.), Morgan Kaufmann, 2001, pp 702-709.
- Werfel, Justin, Mitchell, Melanie, and Crutchfield, James P. "Resource Sharing and Coevolution in Evolving Cellular Automata." *IEEE Trans. Evol. Comp.* 4(4), 388-393, 1999.