Student Papers
Evolutionary Computation Class
Spring 2007

Edited by
Ivan Garibay

Technical Report Number CS-TR-07-11
School of Electrical Engineering and Computer Science
University of Central Florida, November, 2007

Preface

These proceedings contain the student papers presented as final projects for the Evolutionary Computation class (CAP 5512) that I taught at the University of Central Florida the Spring of 2007. All papers in this collection present original research in the area of Evolutionary Computation developed during the course of this class, except for two. These are a literature review by Das and a reproduction of results by McDaniel.

More information about this class can be found at
http://ivan.research.ucf.edu/classes/CAP5512_Spring2007/index.htm

Ivan Garibay
Orlando, November 2007

Focused Nash Memory for Coevolution

Kevin M. Kelly
University of Central Florida
CAP 5512
Spring 2007

kkelly_1138@yahoo.com

ABSTRACT

Coevolutionary algorithms have various problems when used. One of these known problems is that of forgetting. This is especially true for intransitive problems. Sevan G. Ficici [1] proposed a memory mechanism to solve this problem using the game theory principle of Nash Equilibrium. This paper expands on that work and looks at ways to focus the "Nash Memory" mechanism as well as exploring it's uses with other game types.

1.INTRODUCTION

The use of game theory principles to assist coevolution has become popular in recent years. It is seen as a way to help solve some of the problems traditionally associated with coevolution (specifically competitive coevolution.)

2.GAME THEORY IN COEVOLUTION

2.1 Problems in Coevolution

When people first started to look into competitive coevolution, it was seen as something of a silver bullet to solve evolutionary algorithms. One problem evolutionary algorithms has always had was in defining the *fitness function*. Coevolution promised to have an ever adapting fitness function that didn't need to be defined specifically. By *competing* with the others in the population (or another population), the system could generate better and better fitness functions automatically and create something of an *arms race*. Unfortunately, the early optimism was replaced with frustration as common problems began to develop.

One of these things is known as the "Red Queen Effect." This describes a situation where an individuals objective fitness (fitness across the broader fitness landscape) could increase, but in the ever changing fitness environment the individual finds itself in, that doesn't actually improve it's chances for selection.

Another problem in coevolution is known as mediocre stable states. This is a problem that occurs when there is a lack of definition or driving force. It is often an unexpected state where the individuals in the population get to a position where moving from their causes problems, and so it becomes stable, but it is an undesired state.

Cyclic dynamics is another typical coevolution problem. Cyclic dynamics are a direct result of the dynamic nature of the fitness landscape. At times, one strategy is good, but as that becomes dominant, a different strategy becomes better. As that strategy becomes dominant, the original strategy could become dominant again causing a non-stop cycle of strategies with no ability to search for better ones.

Related to cyclic dynamics is the principle of forgetting. As the fitness landscape changes, traits that may be needed later are lost. This could be from cyclic dynamics or drift. Drift is when a trait does not distinguish it from a fitness perspective from other individuals and therefore can stay or be removed without any pressure from the system. The problem comes when this trait is then needed later and needed to be completely relearned.

2.2 Game Theory

Game theory recently has become popular to address some of the issues in coevolution because the same problems exist in game theory. Two concepts in game theory that have been looked into the most are Pareto Dominance and Nash Equilibrium [9].

Pareto Dominance defines the dominance of one strategy over all others for some condition. The set of all undominated strategies is referred to as the Pareto Front. This front gives the possible trade-offs in strategies. All strategies on the Pareto Front are the best given some condition.

This is the currently most popular mechanism in applying game theory to competitive coevolution. Ficici [2] describes a method of applying Pareto Dominance to coevolution. The population evolves against the current dominant set rather than each other. This gives a clear goal to the evolution as helps solve problems such as the Red Queen effect as well as mediocre stable states. It also has some effect on intransitive cycles, however, it doesn't work all that well when those cycles exist. It is able to give an indication of them.

Bucci [4] takes a similar approach in using a Pareto Hill Climbing algorithm (similar approach to Pareto dominance as [2]). He runs experiments using two versions of the Numbers Game (see below for details). He uses the Intransitive Numbers Game as well as the Focusing Game. the intransitive game (as the name implies) has many intransitive cycles. The focusing game does not have intransitive cycles but good competing strategies can be very different (causing forgetting problems). He runs them using the Pareto Hill Climber as well as an algorithm that doesn't use Pareto dominance and shows that the algorithm not using Pareto has significant problems where the Pareto algorithm performs very well.

Bucci [6] formalizes Pareto coevolution for two player, two outcome games. It is typical of most of the work done in game theory for coevolution that they use two player, two outcome, and even zero sum games. Zero sum simply means that the scores of the players (two players in all of these cases) sum to zero. This is usually done for two player games to mean one player wins, and

the other loses. It often also allows each to score zero – meaning a tie.

De Jong in [3] introduces the MaxSolve Algorithm. This paper is often cited by others in the field. This again uses Pareto Dominance as a solution concept. It also looks briefly at Nash Equilibrium, but does not use it much. MaxSolve formalizes an algorithm to use Pareto Equivalence Sets to solve simultaneous goals. It uses the Compare on One Game to test this (see below for details).

De Jong in [5] specifically looks at using Pareto Coevolution solve intransitive games. He claims (as others do) that intransitive games lead to cycling. His claim is that Pareto Coevolution can transform an intransitive game into a transitive one. He tests this using the Intransitive Numbers game.

Another game theory concept often referred to in Nash Equilibrium. This is the concept that says that games can get into a state where a change in strategy for either player would cause their score to go down. This is seen as the optimal strategy for a *rational* player. The problem with Nash Equilibrium is that for a given game, there could be many Nash Equilibrium. This is typically solved in the current research by choosing games with a single Nash Equilibrium as well as using only two-player zero sum games. For two-player zero sum games, all Nash Equilibrium are the same total fitness. For certain cooperative games (such as the Prisoner's Dilemma – which is not zero sum) this is not the case.

Although various authors look briefly at Nash Equilibrium as a solution concept, the only one that goes deeply into using it is Ficici [1]. It is used there as the goal for a memory mechanism to avoid forgetting and intransitive states. This paper looks more deeply into that.

3. Numbers Game

The Numbers Game really a loosely related set of games that are very good for looking into evolutionary strategies. A Number Game is usually defined by each player playing an *n*-dimentinal vector of integers. The specific game defines which strategy wins.

Numbers Games are popular in coevolution research because they can be simply evolved and can be defined to create many of the problems where research is desired. Three different numbers games are described below that have been used in coevolution research.

3.1 Intransitive Numbers Game

The Intransitive Numbers Game was first defined by Watson and Pollack [7]. It was created to make intransitive cycles. An intransitive cycle is where strategy A beats strategy B, B beats strategy C, but C beats strategy A. The most obvious example of this is the game of Rock – Paper – Scissors. Rock beats Scissors, Scissors beat Paper, and Paper beats Rock.

For a vector of *n* dimensions, the winner of the intransitive numbers game can be simply defined as the the highest value in the dimension where the competing strategies are closest. Ficici[1] also adds a value to it defining the minimum distance away it will consider for closeness. This can allow it to be set up so that a dimension will not be declared unless it is a certain value different. The significance of this will be described more later.

3.2 Focussing Game

The Focussing Game is used by Bucci[4]. This defines a game that creates asymmetric winning strategies. That is where two equally good strategies could be very different. The Focussing Game simply takes the highest of all dimensions and compares those values.

For a vector of *n* dimensions, the winner of the focusing game is the one with the highest dimension in *any* dimension. This causes a focusing problem for coevolution because it works best with a diverse population but can cause a population to focus to much on one dimension.

3.3 Compare on One

The Compare on One game is used in De Jong[3]. It describes a game similar to the Focussing game. Instead of allowing the highest in any dimension to be compared though, it compares a particular dimension based on the highest dimension of the *test*. In this, the strategies that will do the best overall are the ones that have a high value in *all* dimensions but if the tests are all in a single dimension, a coevolution strategy could evolve only strategies in that dimension again causing a focusing problem.

Table 1. Numbers Game Properies

Game Type	Definition	Property
Intransitive Numbers Game	Highest in closest dimension	Intransitive sets
Focussing Game	Highest in *any* dimension	Asymmetric winning strategies
Compare on One	Highest in the dimension of the test set	Causes focus problems

4. Nash Memory

Ficici[1] the uses the concept of Nash Equilibrium as a structure for memory management to improve the process of coevolution. He attempts to solve the problems of forgetting and cycling in coevolution by adding a solution concept and memory mechanism. He more or less succeeds but with some practical problems.

Memory is needed in coevolution to help solve the problem of forgetting in these algorithms. That is while searching the evolutionary space, traits that were useful at one point are lost but are needed later. This can happen because for a given environment, that trait is not currently useful or because it simply migrates with similarly useful traits. Because the search space may be volatile (such as in a game) the heuristic that determines if it is good or not needs to adjust as well. This is the case in such things as rock-paper-scissors where the best strategy is dependent on what your opponent plays.

Nash Equilibrium helps to deal with this volatile environment. Nash Equilibrium says that there is a strategy that is the best strategy to play against itself. That strategy could be a mixed strategy meaning that there is a set of strategies that are played at some probability. For example, in rock-papers-scissors, rock, paper, and scissors are useful at equal probabilities. Therefore the mixed strategy is to play each one at that probability. Each of those pure strategies (i.e. rock, paper, and scissors) are said to support the mixed strategy. If a strategy is a Nash Equilibrium, it also beats all pure strategies. This gives a good goal state as well as allows the goal state to encode the possibility to play multiple strategies because of different environments. It should be pointed out that this only works when all of the pure strategies can be defined and finite.

One problem seen in the experiments this paper were that the Nash Approximation grows significantly over time. It creates such a complex system that it can cause serious practical problems. This problem will be looked at in more detail as well as a possible solution to the problem – Focused Nash Memory.

4.1 Description

The concept behind Nash Memory is to use two sets of strategy to maintain the memory and a heuristic mechanism (in this case an evolutionary algorithm) to produce a set of strategies that can beat the currently accepted bests strategy.

This uses the property of Nash Equilibrium that for zero-sum games, the Nash Equilibrium is secure (i.e. can't be beat) against all non-Nash strategies. It also uses the property that a Nash strategy can be defined as a *mixed strategy*.

A *pure strategy* is as a single strategy of the game. In this case that would be a single vector. A *mixed strategy* is a set of pure strategies and a value from 0 to 1 for each of those strategies defining the ratio for use of that strategy.

The memory of Nash Memory consists of the current best Nash Approximation and a Memory set of all pure strategies that were part of the the Nash Approximation at some time.

A set of all winners of the heuristic set (pure strategies that score greater than zero against the current Nash Approximation) is delivered by evolutionary algorithm. Those winners, the current Nash Approximation, and all of the strategies in Memory are put into a game matrix and solved with Linear Programming. The linear programming used is described below. This provides back a set of strategies supporting the new Nash Approximation and the ratios for that mixed set. All of the strategies that were in the Nash Approximation (or already in memory) but no longer are, are added to the memory set for future use.

This memory mechanism allows the system to bring back from the dead strategies that were once considered good. It breaks intransitive cycles by allowing all strategies in the cycle to be included in the Nash Approximation and as better strategies are found, those can get replaced. It should be noted that a pure set is simply a special mixed set which only has a single strategy whose use value is 1.0.

4.2 Linear Programming (SIMPLEX)

The Linear Programming used is known as the SIMPLEX algorithm[8]. SIMPLEX takes a game defined as a payoff matrix (i.e. for each strategy the matrix defines it's score against each other strategy). A row is added to the end of the matrix, known as the *objective entries*, representing the percent played for each of the row player's strategies (initialized to -1) and a final column representing the percent played for the column player's strategies (initialized to 1). It also then adds a corner value which represents the maximum value of the function. For games solving, this should eventually approach one.

The pseudo-code to the simplex algorithm is as follows:

1. Are any negative values in the objective entries
    1. no – solution found
2. Select a negative objective entry
3. In the column above the selection, select the row with a value that minimizes the ratio (final column value) / (row value) where row value != zero (known as the ɵ-ratio.)
4. Pivot the table at that selected row and column to obtain the new tableau. (modifies all values, making the objective entry non-negative)

This algorithm finishes with a set of pivoted algorithms and values for each of those algorithms. If the algorithms were not pivoted, or their value is ~0 then it is considered "not-used" for purposes of Nash Memory. The percent values used are treated as the values for the mixed sets.

5.Experimentation Tool and Software Design

To conduct the experiments for this project, a test tool was developed. This tool has a GUI interface and was developed using Object Oriented techniques to allow easy experimentation and extension. One of the goals for this was to allow easy addition of different types of games and evolutionary algorithm techniques.

When developing the experimentation tool, it was clear that SIMPLEX does not deal well with very small or very large values. That is to say, the mathematics of it work fine, but practically speaking it starts to fail. The precision of the double values used in implementation are so great that values that should be 0 (and therefore ignored at certain parts of the algorithm) are instead used and create equally large numbers in the rotation. It was therefore necessary to stop the calculations ar a certain number of significant digits.

6. Experiments

The first experiments done were to test the ability of Nash Memory as described by Ficici[1]. The system was run using the Intransitive Numbers Game with 2-dimensional vectors. The values of each dimension were from 0-100. They were represented as a 200 bit binary string with the value of the first dimension being the sum of bits 0-99 and the value of the second dimension being the sum of bits 100-199. It was run in epochs of 30 generations each. At the end of each epoch, the *best value only* was taken and compared to the current Nash Approximation. If it dominated the Nash Approximation (i.e. it at least tied every pure strategy in Nash and beat at least one of them) then that

strategy became the new Nash Approximation (with mixed value of 1.0) and the entire previous Nash Approximation was added to Memory. If the best strategy at the end of the epoch scored less than 0.0, the Nash Approximation was not updated. Otherwise, SIMPLEX was run against the best strategy, the current Nash Approximation, and the Memory. All used strategies with a use percent greater than zero become the new Nash Approximation, and all strategies previously in Nash or Memory that were not used were put into Memory.

The GA used only mutation (no crossover) with a per-bit mutation rate of 0.01. It used Elitism of 10 and Tournament Selection (tournament size of 2).

The results obtained were not quite as clean as those presented in [1], but did show the same trends. Figure 1 show the mean over 20 runs of the best scores against the Nash Approximation at the end of each epoch. It shows, like Ficici[1] did that early on, it was easy to obtain a good score against The approximation, but as the approximation became better, it was more difficult to obtain a good score against it.

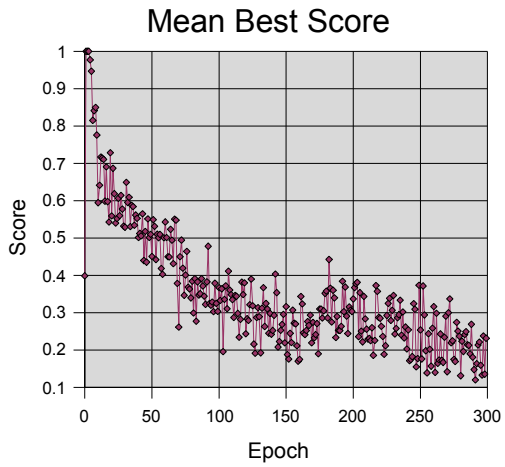


Figure 1: Mean best score against Nash Approximation at each epoch

What is not completely clear is whether it is the memory mechanism that causes this, or if it is just a lack of ability of the GA. Although it becomes more difficult to find a good score against the Nash Approximation, the approximatin is not really very close to the actual Nash value of the game (100, 100). The mixed set is typically supported at the end with strategies such as (70, 48) and (69, 72).

Figure 2 shows the mean of the average population score against the Nash Approximation. This shows that although the GA can produce an individual that can usually beat the Nash Approximation (though just slightly) the Nash Approximation does better against the averge.

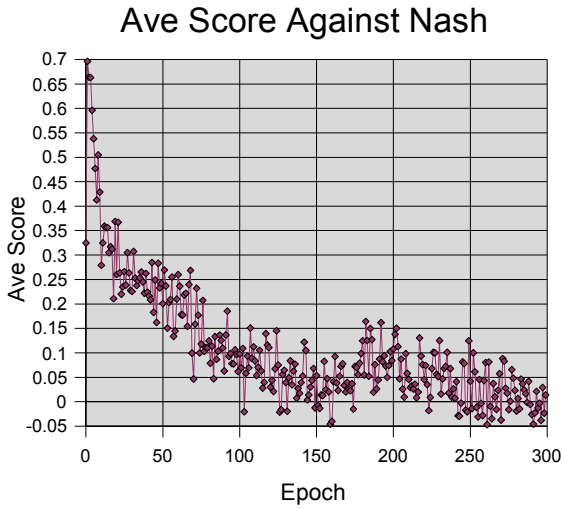


Figure 2: Average Population Score Against Nash Approximation

One notable trend in the system is that as the system runs, the number of strategies in support of the Nash Approximation gets very large. Early on, it can actually drop back down to 1 as the system finds a dominant strategy, but later, it becomes large. The graph in Figure 3 shows the mean number of strategies in support of the Nash Approximation at the end of each epoch.

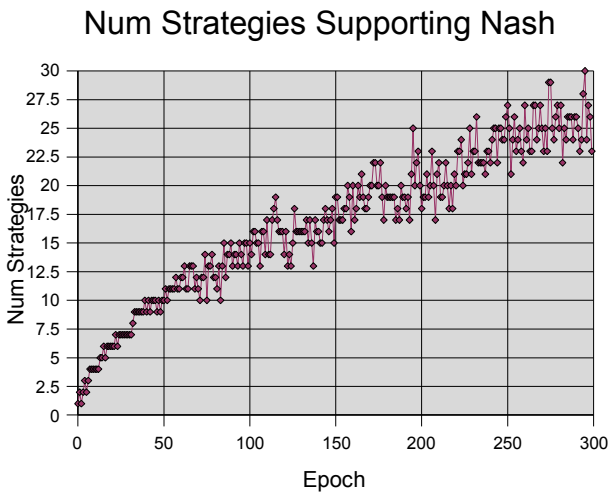


Figure 3: Number of strategies in support of Nash Approximation at each Epoch

The number continues to grow. This is actually much smaller than what Ficici[1] had in his results. For a system with an actual equilibrium of a single pure strategy, it should eventually get smaller but doesn't. As an attempt to improve the GA and hopefully get better Nash Approximations, the experiments were run again but with different values for the GA,

For the second run, it was attempted to converge more quickly to better obtain values that could beat the Nash Approximation. To do this, the mutation rate was lowered to 0.001 and two point crossover was added with a rate of 0.9. It still used a population

of 100, number of generations per epoch of 30, and tournament selection with size 2. The results of those are shown in Figure 4 and 5 below.

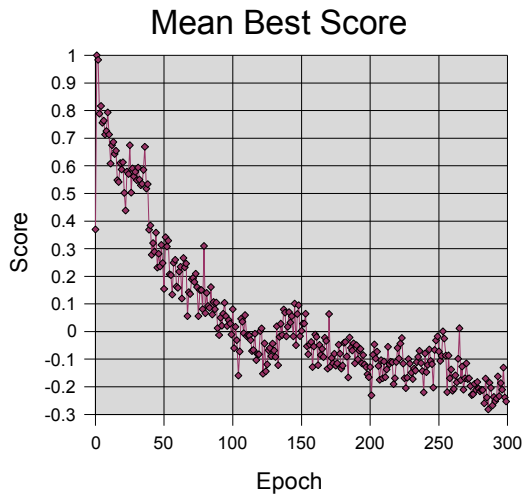


Figure 4: Mean Best score against Nash over 20 runs (xover 0.9, mut 0.001)

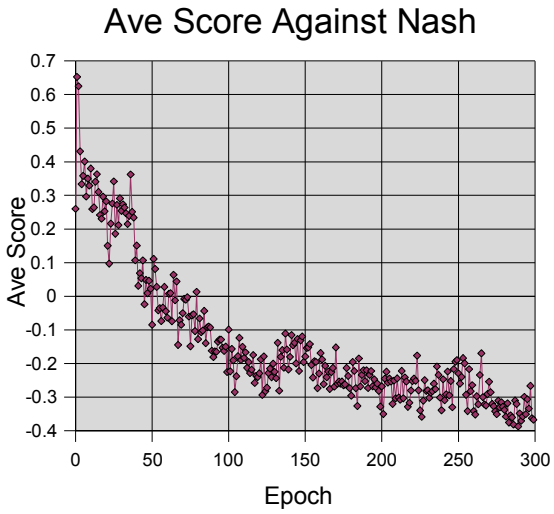


Figure 5: Average Population Score against Nash

It is interesting that the GA much more quickly fails to produce strategies that can beat the Nash Approximation. In fact, there were some runs that didn't produce another strategy that could win after the 70th or 80th epoch. The final strategies are still as good or better than those produced with the weaker GA. For instance, one run got a final Nash Approximation of two strategies (71, 48) – played at 0.33 and (70, 79) – played at 0.66. That is still not all that close to (100, 100) but is still fairly relatively good.

With a stronger GA the average number of stragegies needed went down significantly as well. Figure 6 shows the graph of number of strategies at each epoch.

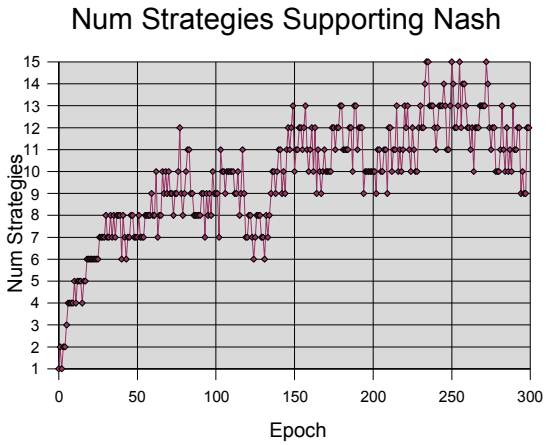


Figure 6: Strategies supporting nash from run 2 (xover 0.9, mut 0.001)

The number of strategies in support even appears to go down as the system gets better and better approximations.

7.Focused Nash Memory

The number of strategies needed to support the nash approximation takes a disturbing trend in the normal case. As higher epochs are used it keeps greater and greater numbers of strategies in support. In Ficici's original paper [1] he uses a strategy to finally get a very close approximation of the actual Nash Equilibrium of (100, 100) but it required 123 strategies in support [1].

The growing number of strategies presents a fairly harsh problem for practical use of this memory mechanism. With greater numbers in the Nash Approximation, all other calculations get greater.

The problem with the growing number of strategies in support of the Nash Approximation has to do with the SIMPLEX algorithm itself. SIMPLEX will keep strategies that have a very very low actual play value. If the mixed values are studied on a standard run with a lot of strategies in support, it is observed that there are typically a few high use strategies and a significant number of very low used strategies (see Table 2). These strategies contribute very little to the actual score of the Nash Approximation.

Table 2. Example of a Nash Approximation

Strategy	Use	Strategy	Use
(55, 35)	0.0002	(58, 42)	0.09909
(55, 59)	0.00145	(33, 62)	0.12329
(64, 36)	0.00497	(58, 53)	0.00439

(52, 58)	0.00439	(52, 58)	0.0138
(54, 53)	0.03593	(61, 41)	0.01152
(65, 38)	0.00374	(71, 67)	0.18891
(63, 35)	0.01086	(73, 41)	0.03284
(46, 62)	0.02179	(57, 54)	0.01439
(62, 46)	0.07393	(56, 51)	0.03884
(56, 51)	0.03884	(62, 52)	0.02115
(52, 55)	0.01843	(61, 38)	0.0165
(65, 53)	0.02179	(51, 71)	0.11087
(53, 46)	0.01902	(45, 71)	0.06515
(27, 57)	0.03517	(67, 35)	0.01198

You can see in Table 2 that there are only a few strategies even above 0.1. Looking at two of those strategies (71, 67) and (51, 71) they are clearly dominant over most of the other strategies in the list. The question then becomes, what benefit does the system gain by keeping these low use strategies to evolve against.

In order to focus the evolution of the Nash Approximation, a *Focus Value* was added to the experiment. This value determines the lowest use percent that would be allowed to support the Nash Approximation. This was added to the end of the SIMPLEX algorithm so that any strategy that wasn't at least that good, was not allowed in the Nash Approximation. The experiments from earlier were then run with only that change. The focus value selected for this experiment was 0.05. The Figure 7 shows the mean value of the best score obtained with that focus value on the first experiment (as Figure 1 above).

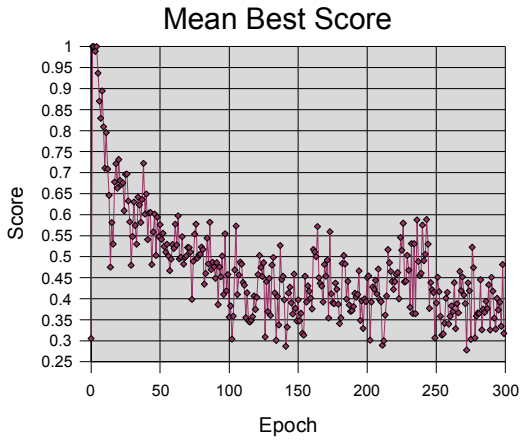


Figure 7: Mean Best Score with Focus Value 0.05

What is seen is that the Nash Approximation has a much harder time beating the best the heuristic can come up with. Like before, this is only half of the story. It is unclear if this is because it the memory is failing or the GA if just doing better. Observing the approximations obtained look at least as good, if not better than those obtained without the focus.

One thing that is clear is that the size of the Nash Approximation is significantly lower. Figure 8 shows the number of strategies in support at each epoch.

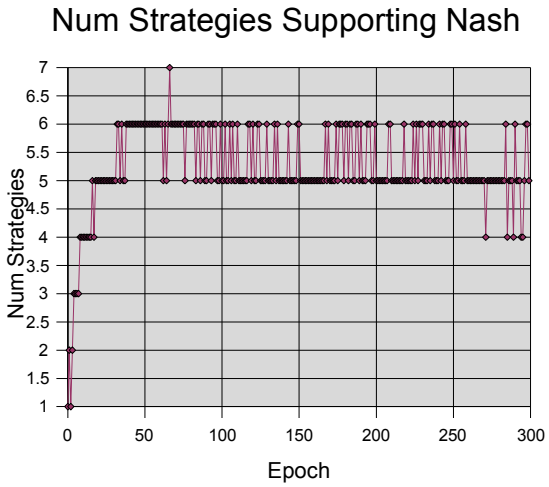


Figure 8: Average Number of strategies in support for Focus Value 0.05

Similar results were found for the second experiment. The real test though on whether or not this is an effective strategy is how good the strategies is produces are. A final set of experiments were run to actually play the approximations against each other at the end of each epoch. The results are show in figure 9.

This experiment averaged the score per epoch over 20 runs. One was run using the a focus value of 0.05 the other with no focus value. The graph shows that the focused Nash memory starts off slightly better but slowly gets worse at higher epochs. This is the opposite of what was expected. It was thought that at higher epochs when the non-focused version got large numbers of strategies, it would become harder to develop strategies and therefor suffer. It appear that the focused version gives the Nash a fast start – ignoring the weaker strategies, but as the non-focused one is able to take more strategies in, it makes up ground.

In any case, the differences in the strategies at any level are fairly small. This demonstrates a trade-off that can be made. There are inherent advantages to having fewer strategies in support (performance, readability, etc) but that can be gained at a slight cost in overall strategy effectivness.

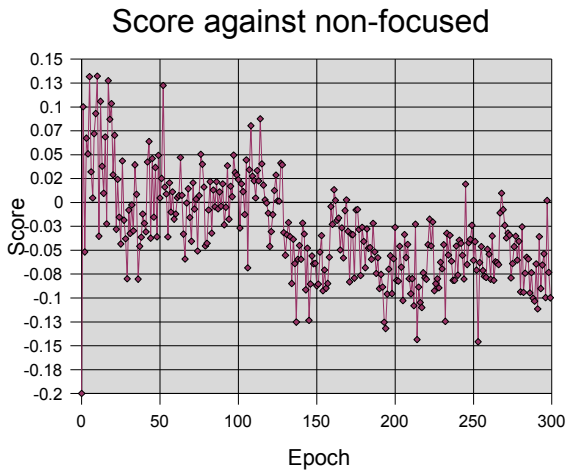


Figure 9: Score of Focused (0.05) against non-focused (Avg over 20 runs)

8.Conclusions and Future Work

Overall the focused Nash Memory strategy did not have the effect that was hoped for. It is possible that different focus values would give better results. It was promising that the negative effect of removing strategies did not cause significant loss of strategy ability. The attempt taken to focus the Nash Memory and reduce the unnecessary support strategies is still a valid mechanism. Instead of the expected utility of increasing the ability to find better strategies, it still has utility in improving performance, at least to the point of justifying more research.

There are other possible ways to focus the ability of Nash Memory that were not looked at here. Instead of taking a pure focus percent of overall strategy use (i.e. 0.05 an better are kept), it way work better to allow it to be a percent of the best strategy. This would allow the focus value to change depending on other strategies present. The purpose is to filter the states when there are a couple really useful strategies and many non-useful strategies. It way also work better (but with a loss of performance) to put the winning strategies that pass through the focus filter back through the SIMPLEX process again. This may work better than the simple normalizing that was used in these experiments. Some of the strategies taken were taken partially because of their ability to beat the strategies that were later removed.

Overall the Nash Memory mechanism helps coevolution avoid some of the problem states. The ability to focus it is needed in some manner to help reduce the bloat of strategies it faces, though it may come at a decrease in overall strategy performance. This is just one more trade off that can be weighed when choosing the algorithm to use.

9.REFERENCES

[1] Ficici, S.G. and J.B. Pollack. A Game-Theoretic Memory Mechanism for Coevolution. *2003 Genetic and Evolutionary Computation Conference*, 286-297. Springer, 2003.

[2] Ficici, S.G. and J.B. Pollack. A Game-Theoretic Approach to the Simple Coevolutionary Algorithm. *Parallel Problem Solving from Nature VI*, 467-476, Springer, 2000

[3] de Jong, Edwin. The MaxSolve Algorithm for Coevolution. *GECCO 2005, June 25-29.*

[4] Bucci, A. Pollack, J.B. Focusing verses Intransitivity. *GECCO 2003,* 250-261

[5] de Jong, E.D. Intransitivity in Coevolution, *Lecture Notes in Computer Science,* 2004, 843-851, Spinger-Verlag.

[6] Bucci, Anthony, Jordan Pollack, A Mathematical Framework for the Study of Coevolution. *FOGA 7: Proceedings of the Foundations of Genetic Algorithms Workshop*, San Francisco, CA, Mogam Kaugmann Publishers (2003) 221-235.

[7] Watson, R.A, J.B. Pollack. Coevolutionary Dynamics in a Minimal Substrate. *GECCO 2001.* 702-709. Morgan Kaufmann, 2001.

[8] Brickman, Louse, *Mathematical Introduction to Linear Programming and Game Theory*. 1989, Springer-Velag, New York, NY.

[9] Dutta, Prajit K. *Strategies and Game, Theory and Practice*. The MIT Press, Camridge Massachusetts, 1999.

Examining Cooperative Co-evolution and Collaboration Methods on Deceptive Landscapes

Phillip Verbancsics
University of Central Florida
School of EECS
Orlando, FL 32826
407-595-8873, 1
verb@cs.ucf.edu

ABSTRACT

Cooperative Co-evolution generates interest with its ability to solve complex domains by breaking down the problem into subcomponents. To be used to the best effect, we must understand the performance of cooperative co-evolution under varying circumstances. This paper aims to further the understanding the capabilities of cooperative co-evolution by expanding the understanding of the effect of collaboration methods and extending the analysis to deceptive domains. We show that cooperative co-evolution is fully capable of handling deceptive domains through the variation of collaboration methods.

Categories and Subject Descriptors

I.2.m [**Artificial Intelligence**]: Evolutionary Computation

General Terms

Algorithms.

Keywords

cooperative co-evolution, collaboration methods, deceptive landscapes, performance

1. Introduction

Understanding is the key for the proper use of any technology. Evolutionary Algorithms (EAs), unaltered, provide relatively powerful methods for problem solving and function optimization. The dynamics underlying the basic EA model has been studied since the days of their creation [5]. EAs also face difficulties in the face of certain domains of fitness landscapes that include deception or reduce the effectiveness of search down to random [3]. Further research discovered ways to compensate for these deficiencies, however knowledge of the problem must first exist before a solution can be crafted [4][11].

More recently, attention has shifted to further extensions of the basic EA model, such as generative systems and co-evolutionary systems. Co-evolutionary algorithms can be further broken down into competitive and cooperative. Competitive focuses on ratcheting up the arms race between individuals to prevent stagnation and to provide better solutions under conditions where the optimal may be unknown or unclear. Cooperative focuses more on decomposing a problem into subcomponents that then can be optimized separately and later combined together to collaboratively find a solution [1].

In this paper, we are concerned with the use of cooperative co-evolutionary algorithms (CCEAs) as static function optimizers, as introduced in [10]. Similar to basic EAs, CCEAs have dynamics that must be understood to be used effectively. The unique features of these collaborative systems were explored in [12]. These features include frequency of interaction and collaboration schemes. In this paper, we will be examining the effect of varying collaboration schemes. Further work has been done in understanding the actual behavior and dynamics of CCEAs in [6][7][8][9][12][13][14]. CCEAs also have their own class of problematic dynamics, as explored in [2]. These involve the devolvement to sub-optimal stable states from which escape is difficult. The dynamics of CCEAs on these types of landscapes were explored in [9].

This paper will further expand on the dynamical understandings of the behavior of CCEAs on particular landscapes, especially in reference to the variation of collaboration methods. We will be varying two of three collaboration parameters described in [12]. The three parameters are number of collaborators, selection pressure on collaborators and fitness assignment method. We will focus on the first two and their effects on landscapes with deceptive features. Deceptive landscapes have been shown to be difficult to overcome for EAs [3]. We will examine the dynamics of CCEAs on deceptive landscapes adapted from [9]. Landscapes with stables states provide difficulty for CCEAs, but can be overcome with different collaboration methods [6]. Adding deceptiveness, these landscapes will provide insight into problems which are not just difficult for CCEAs or EAs, but provides difficulty for both. We will show that the variance of collaboration methods can provide capability for overcoming deception, while balancing out the capability of performance on non-deceptive landscapes. This provides insight into parameter settings that can be used if the CCEA user is unsure whether their landscape contains deceptive features or not.

2. Experimental Setup

2.1 Algorithm

We use a two population CCEA setup to evolve solutions to two parameters in a fitness equation. The basic algorithm first initializes the two populations with random real numbers. Using *sequential update timing* [12], each population takes turns being evaluated, switching with one turn intervals. During a populations turn, it is evaluated using collaborators from the other population, selected using tournament size 2 and mutated using a Gaussian distribution. Additionally, collaborators are selected at the end of the turn to be used with the other population. The parameters used for all runs remained the same, with the exception of number of collaborators, collaborator selection pressure and deceptiveness (See Table 1).

Table 1. Parameters for CCEA

Parameter	Value
Individuals per Population	10
Evaluations per Run	1000
Selection Rate	1.0
Selection Method	Tournament, Size 2
Elitism	1
Mutation Rate	0.75
Gaussian Sigma	0.25
Number of Collaborators	1 through 5
Elitism for Collaborators	0 through 5
Fitness Assignment Method	Optimistic
Deceptiveness	0.0; 0.9
Maximum Value	8.0

Following from [9] we decided to maintain a small population for better effects of overcoming the problems inherent in some fitness landscapes. The other settings were also transferred over from [9] to better verify the functionality and provide contrast to our extended results. For each experiment, we varied the number of collaborators from 1 to 5, inclusive, allowing us to collaborate with up to half the other population. We varied the selection pressure on the collaborators from 0 to 5, inclusive, where this indicates the number of "best" collaborators that are used for the next collaboration and the rest are chosen randomly with replacement. The final parameter, deceptiveness, controls how deceptive the landscape is. This is further explored when we look at the fitness landscapes. Sufficed to say, it ranges from 0.0 to 1.0 and higher values indicate greater deception while 0.0 is no deception. We use a maximum individual value of 8.0, minimum at 0.0, to provide continuity for results from [9]. Each parameters combination is run a total of fifty times and the results are tabulated from the statistics of the combined runs.

2.2 Fitness Landscapes

There are essentially four landscapes under examination. The first two are landscapes that have been examined before in [9]. These are the One Ridge and Two Ridge landscapes that provided insight into the dynamics of CCEAs (See Figures 1 and 2).

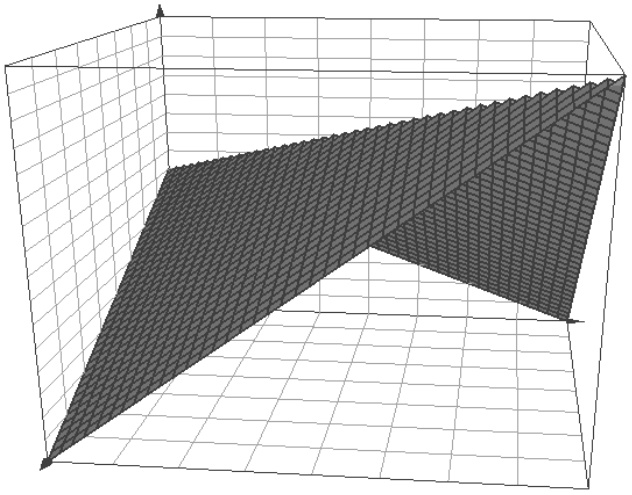


Figure 1. One Ridge with No Deceptiveness, d= 0.0

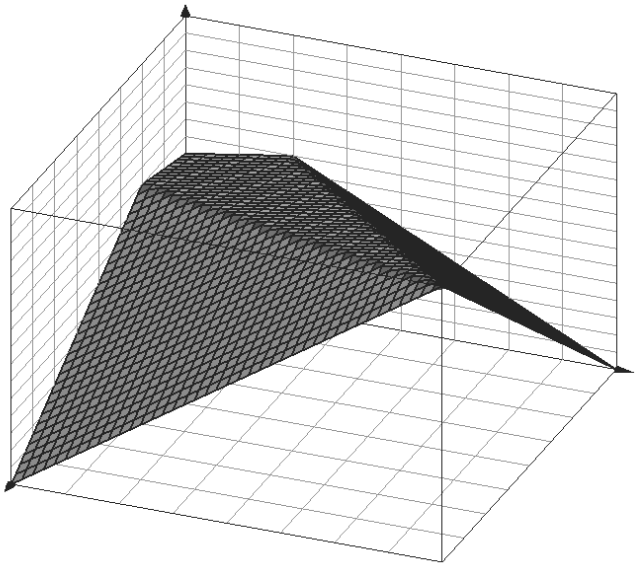


Figure 2. Two Ridge with No Deceptiveness, d = 0.0

The second two are extensions of these basic landscapes which are extended to include deceptive elements. These

deceptive elements provide a secondary hill that leads the away from the global optimum and towards a local optimum. In this case, the One Ridge and Two Ridge have a local optimum at (0,0) decrease in value until a delta point, then increase from that delta point to a global maximum at point (m,m) (See Figures 3 and 4).

The equation that governs the one ridge landscapes is as follows:

$$1(x,y)_{m,d} = \max\begin{pmatrix} m+2*\min(x,y)-\max(x,y), \\ d*m+2*\min(m-x,m-y)-\max(m-x,m-y) \end{pmatrix}$$

In the above equation, m represents the maximum value x and y can take on, while d represents the deceptiveness factor that governs how much of the landscape is deceptive. When d is 1.0, we have two equal length ridge areas, when it is 0.0, the deceptive landscape is completely eliminated. Similarly, the equation for the two ridge landscape is:

$$2(x,y)_{m,d} = \begin{cases} \begin{cases} m+\dfrac{m-3x+4y}{2} \; if\left(y<\dfrac{4x-n}{3}\right) \\ m+\dfrac{x+y}{2} \; if\left(y<\dfrac{3x+n}{4}\right) \\ m+\dfrac{m+4x-3y}{2} \; otherwise \end{cases} if(x+y \geq m*d) \\ \begin{cases} m*d+\dfrac{m-3(m-x)+4(m-y)}{2} \; if\left(m-y<\dfrac{4(m-x)-n}{3}\right) \\ m*d+\dfrac{m-x+m-y}{2} \; if\left(m-y<\dfrac{3(m-x)+n}{4}\right) \\ m*d+\dfrac{m+4(m-x)-3(m-y)}{2} \; otherwise \end{cases} \end{cases}$$

The two ridge function has similar functionality to the previous one ridge function, with m being the maximum value of x and y and d being the deceptiveness factor. Overall, the deceptiveness factor controls the size of the deceptive area and the local optima of the deceptive area (See Figures 3 and 4).

In [9], we saw that the dynamics of the CCEA on the One Ridge and Two Ridge landscapes greatly differed. These dynamics were further explored in [6][7] in reference to both the variation of interaction frequency and collaboration methods. We will see that, while the One Ridge and Two Ridge functions have different effects in the collaboration method performance especially in regards to One Best method, adding deception to either landscape causes similar deception effects on the collaboration methods. This paper examines landscapes with higher deceptiveness to evaluate the performance of the CCEA under harsher conditions.

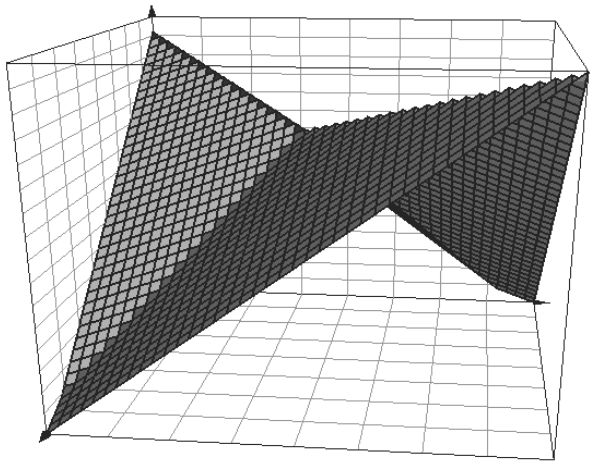


Figure 3. One Ridge Function with Deceptiveness, d = 0.9

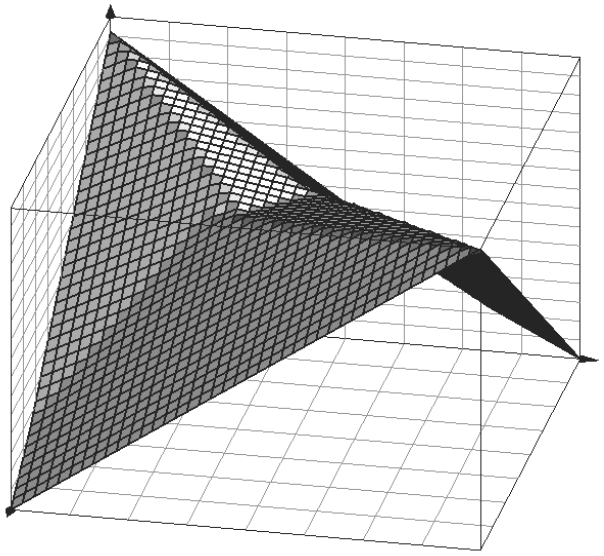


Figure 4. Two Ridge with Deceptiveness, d = 0.9

3. Results

3.1. One Ridge

It is important to establish a baseline for performance of these landscapes before extending them into deceptive landscapes. Without a clear understanding of the initial effects of the variation of collaboration methods on non-deceptive landscapes, its more difficult to interpret the exact effects that varying collaboration methods has on deceptive landscapes. By contrasting with a baseline, we are allowed insight into the dynamics that are unique to the deceptive landscape. The effects of varying collaboration methods are illustrated in figure 5.

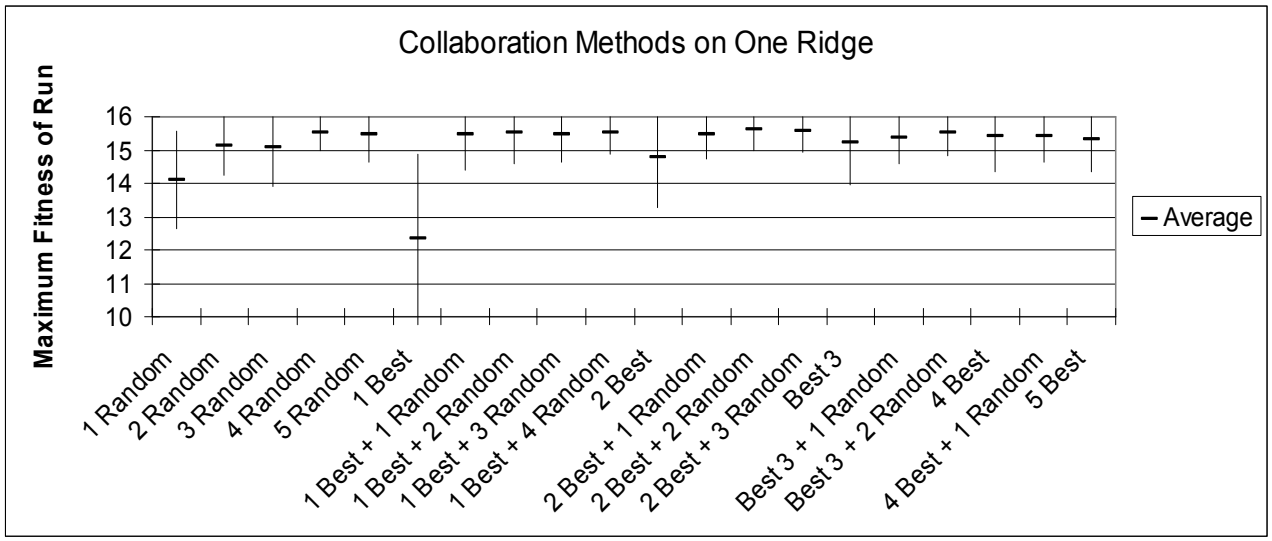


Figure 5. Best of Run Performance for Different Collaboration Methods over fifty runs with a 95% confidence interval on One Ridge function with no deception, Deceptiveness d = 0.0

Figure 5 shows the performance of different collaboration schemes on the One Ridge function. These schemes vary from one collaborator up to five collaborators and a selection pressure on the collaborators from none being elite, meaning all random selection, to complete elitism with all collaborators being selected through elitism. With a maximum value of 8.0 for each individual, the optimal value it 16.0 while the minima are 0.0. The figure shows the average best of run from fifty runs and the 95% confidence interval around the average, showing what best value an estimated 95% of runs can achieve. It is arranged by selection pressure and then number of collaborators. Overall, the best performer is (2 best + 2 random), while the worst is 1 best.

3.2. Deceptive One Ridge

The performance of the different collaboration methods on a one ridge with deception, *d,* equal to 0.9 is shown in figure 6. It is important to keep in mind that while adding deception to the landscape, we also increase the fitness for a portion of the landscape, leading to an increased area of higher fitness versus the same landscape with no deception in it. The organization remains the same as the One Ridge with no deception. The best performer in this case is (1 best + 2 random), while the worst performer remains 1 best. The best of run fitness is not the only important statistic to examine in a deceptive landscape, but also the number of times it is deceived to see how often the search results in an goes in an path that does not lead to the optimum (see Figure 7).

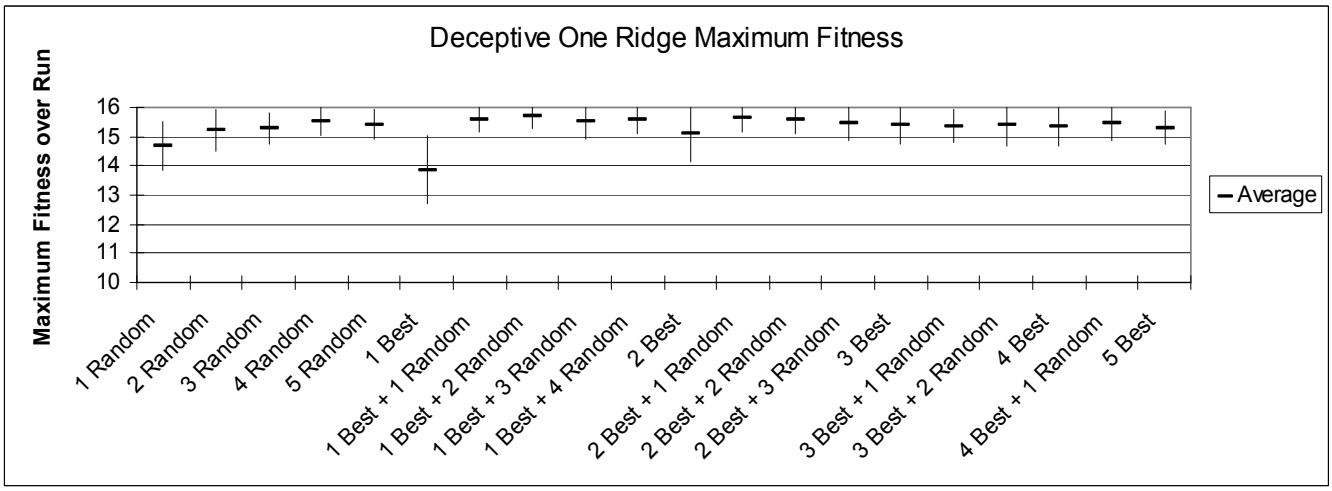


Figure 6. Best of Run Performance for Different Collaboration Methods over fifty runs with a 95% confidence interval on One Ridge function with Deceptiveness, d = 0.9

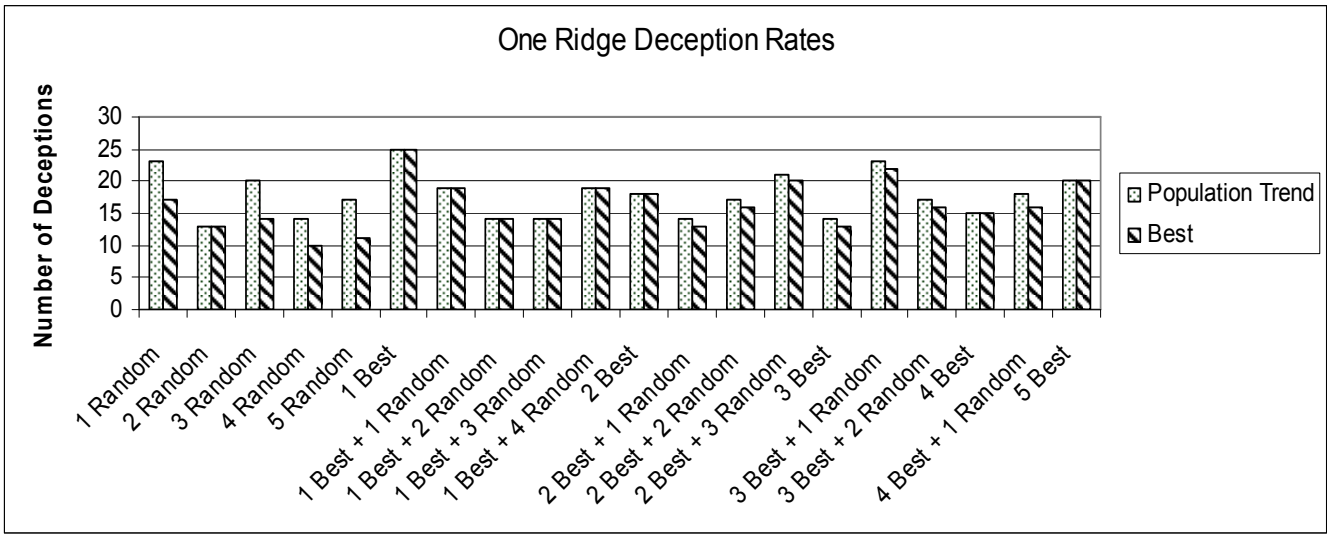


Figure 7. The times deceived out of 50 runs for different collaboration methods on One Ridge with Deceptiveness, d, equal to 0.9. Population trend indicates whether the final generation's best collaboration was in the deceptive region while Best indicates the rate at which the best of the entire run was within the deceptive region.

Figure 7 shows the deception rates over the fifty runs for the One Ridge with deceptiveness, d, equal to 0.9. In it, we examine the number of times, out of the fifty runs, that the best collaboration in the populations in the final generation (e.g. best individual in the population at the end) is on the deceptive area of the landscape and also the number of times the best collaboration (e.g. the global best) of the entire run was on the deceptive landscape area. In both cases the best collaboration represents the best individual, since we are using Optimistic fitness assignment. These give us an idea of how well the collaboration methods are handling the deception. The maximum number of deceptions possible is all fifty of the runs, while the minimum is zero. We can see both values peek at 1 best, while best reaches its minimum at 4 random and the population trend has a minimum at 2 random.

3.3. Two Ridge

As seen in [6][9], the One Ridge function has unique properties in behavior in relation to the behavior of the CCEA. Figure 8 demonstrates the differences that were found. As in the case of the One Ridge function, we see the average best of run fitness with a 95% confidence interval for each of the collaboration methods being looked at. The best performing collaboration method in this case is the 1 best, while the worst performing is the 5 random collaboration method.

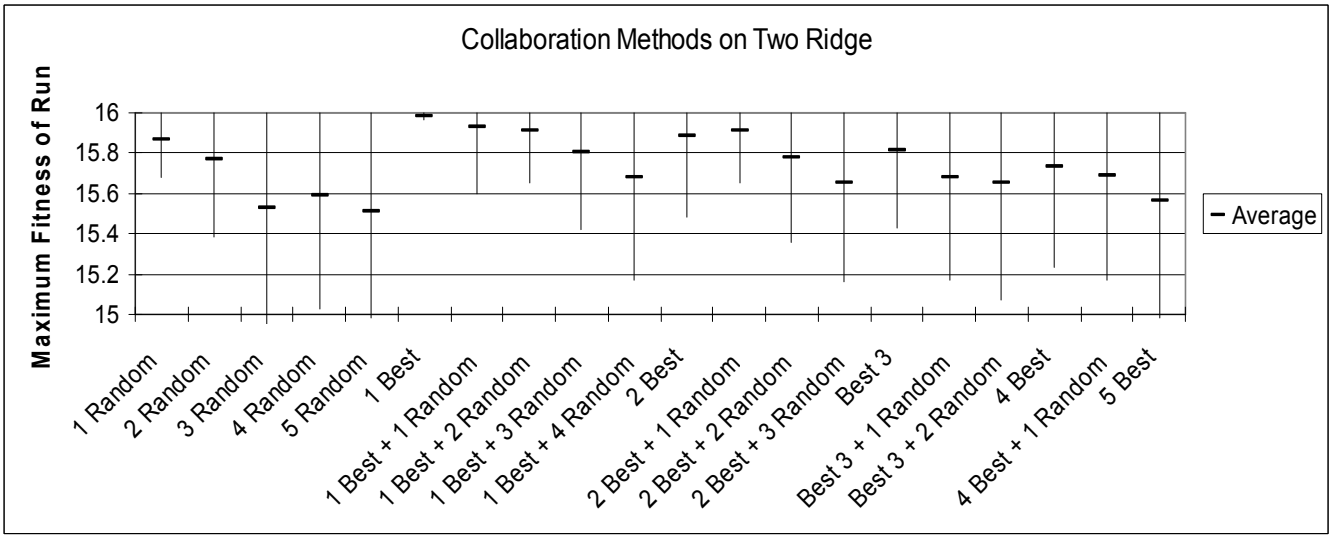


Figure 8. Best of Run Performance for Different Collaboration Methods over fifty runs with a 95% confidence interval on Two Ridge function with no deception, Deceptiveness d = 0.0

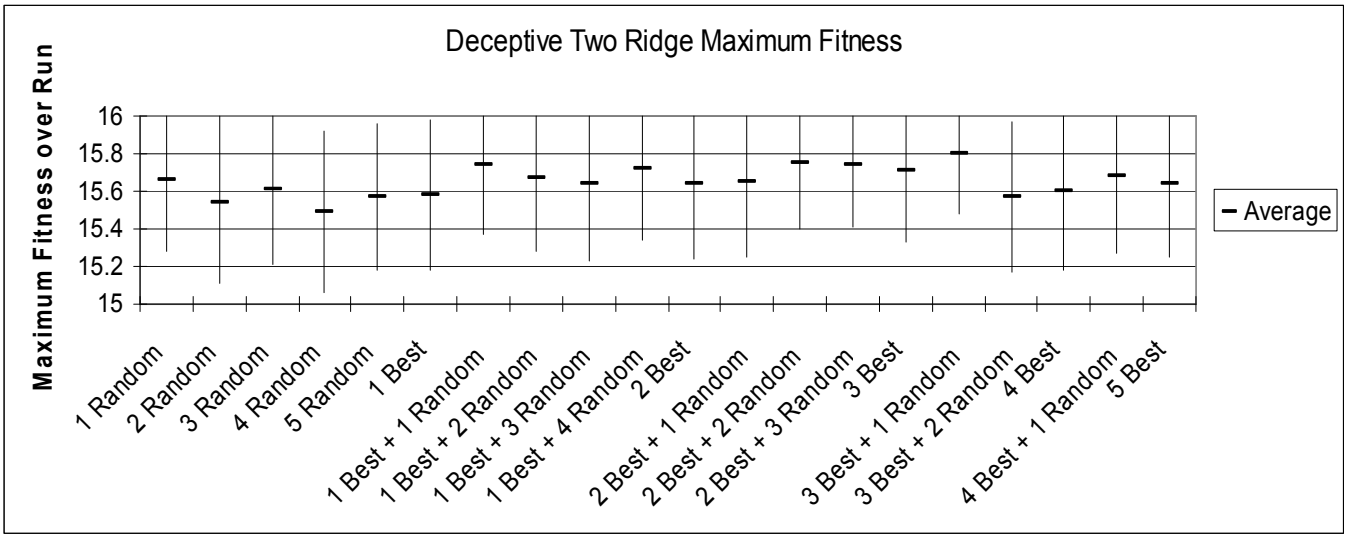


Figure 9. Best of Run Performance for Different Collaboration Methods over fifty runs with a 95% confidence interval on Two Ridge function with Deceptiveness, d = 0.9

3.4 Deceptive Two Ridge

Figure 9 shows the best of run fitness metrics for the deceptive Two Ridge landscape, with the deceptiveness parameter set to 0.9. In this instance, the best collaboration method ended up being the (3 Best + 1 Random), while the worst performer was 4 Random. There is also a greater similarity between the different collaboration methods. This is a change from the non-deceptive Two Ridge function, in which the One Best was the best solution and the various collaboration methods showed a significant difference from each other. In figure 10, we can see that there are several collaboration methods that are equivalent in ability to avoid deception. These include (3 best + 1 random), (2 Best + 3 Random) and (1 Best + 4 Random). Additionally, the 5 Random collaborators method achieves very low best of run deception. The worst performer in this case is the One Best collaboration method. This parallels the result in the deceptive One Ridge function, in which we saw that the One Best collaboration method was the worst.

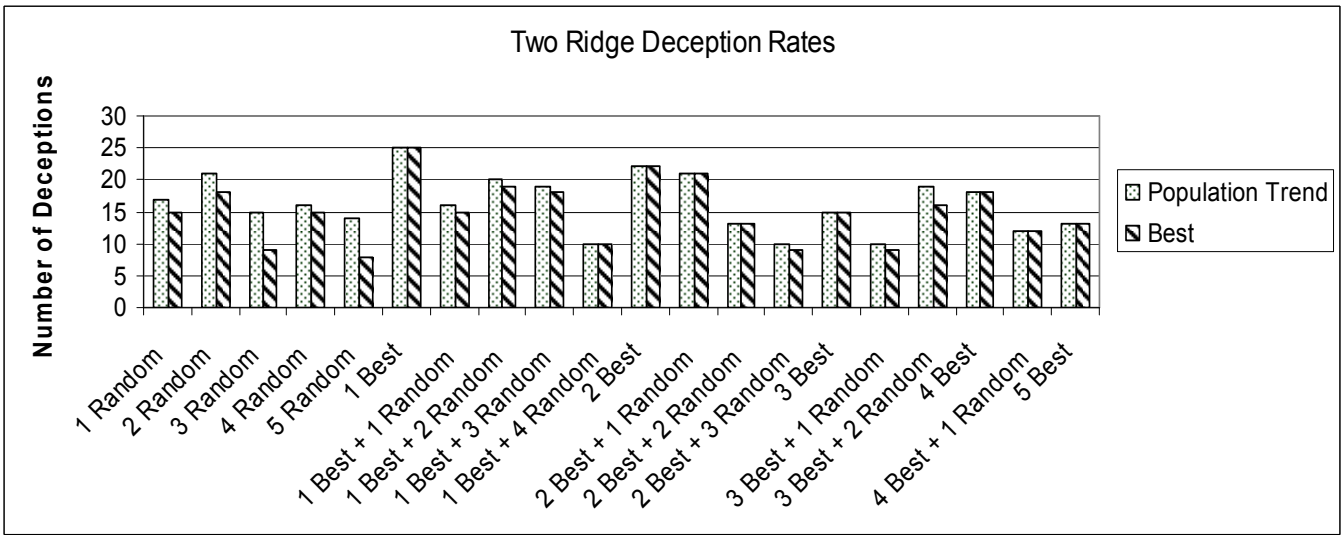


Figure 10. The deception rates for different collaboration methods on Two Ridge with Deceptiveness, d, equal to 0.9. Population trend indicates whether the final generation's best collaboration was in the deceptive region while Best indicates the rate at which the best of the entire run was within the deceptive region

4. Discussion

4.1 Non-deceptive Landscapes

Examining the results from the non-deceptive landscapes, we can see that they confirm the results found in [6]. We can also see some patterns within the individual variations of the parameters. In the case of the One Ridge, we can see that just one collaborator is insufficient for good optimization performance. Both the One Best collaborator and One Random collaborator methods perform worse than all other methods included. This specifically is an artifact of contradictory cross-population epistasis of the One Ridge landscape [6][9]. In Figure 5, we can see that by just increasing the number of collaborators, whether they are random or best collaborators, improves the performance of the optimization. The other trend to note in One Ridge collaboration performances is that pure elitism in collaboration methods also hinders the performance. Even the addition of a single random collaborator increases the ability of the algorithm to deal with the One Ridge landscape. Collaboration techniques with some elitism, those with some number of best collaborators, and some number of random collaborators give us approximately equal good performance.

Moving on to the Two Ridge landscape, with figure 8 we can see a reversal of fortunes for the One Best collaboration method. The One Best method now has exceedingly high performance versus all other collaboration methods. However, all collaboration methods perform very well on this landscape. They are all capable of achieving the maximum value of 16.0, while not going lower than 15.0 in general. Even in this narrow range, there is variation that shows us differing performance capabilities of the collaboration methods. In this landscape, we can see that increasing the number of collaborators actually decreases the performance of the algorithm on the landscape, the opposite of what we saw in the One Ridge landscape. Further, the addition of random collaborators sometimes helps and sometimes hurts, but overall any collaboration method outside One Best and One Random gives worse performance.

Already we can see that the collaboration method chosen biases our algorithm towards certain solutions on the landscape and to overcome obstacles in unknown landscapes, compromise has to be made, following from [15]. Collaboration methods that are not the ideal for either landscape, but still perform well on both give us the capability to make such compromises. In this instance, we can see rather than using the One Best method on the Two Ridge landscape, we can also use the 2 Best and 2 Random collaboration method that gives us good performance on the Two Ridge landscape and also gives us the best performance on the One Ridge landscape. However, the interaction of these settings with the other parameters of the algorithm must be kept in mind.

4.2 Deceptive Landscapes

In the One Ridge Deceptive landscape, figure 6 shows a pattern of performance for collaboration methods very similar to the patterns shown in figure 5, for the One Ridge landscape with no deception in it, keeping in mind that because the deceptive landscape has more of the landscape with higher fitness, the algorithm is achieving better results overall. Again, One Best and One Random show poor performance, while other collaboration methods with more collaborators demonstrate greater capability to handle the One Ridge landscape. The same pattern of increased number of collaborators as well as best collaborators mixed with random demonstrating the best performance appears here. Overall, it seems to demonstrate that deception has no effect on the performance of collaboration methods.

However, let us look at the Deceptive Two Ridge landscape results in figure 9. Again, we must take into account that the landscape is overall higher in fitness and also that the Two Ridge achieves high performance with any collaboration method, so we are examining a much narrower range of performance. Given these caveats, we can still see a marked change from the performance of the collaboration methods on the Deceptive Two Ridge from the same methods performances on the Two Ridge with no deception. We can see that the previous best collaboration method, the One Best collaborator, severely decreased in performance, such that the maximum value isn't even in its 95% confidence interval. Instead, even purely random collaboration methods are outperforming it and instead of increased number of collaborators decreasing performance, as was the case in the Two Ridge with no deception, more collaborators improves the performance over the One Best collaboration method. In this landscape, deceptiveness seems to have a severe effect on the performance of the collaboration methods, to the point that the previous best method is now one of the worst methods. We are now left with two contrasting views, where one shows little to no change in the behavior of the collaboration methods while the other shows great changes. The question is then whether deceptiveness truly has no effect on one and a great effect on the other, or if there is more to the story than the maximum fitness over multiple runs is showing us.

Let us then examine how often the collaboration methods are deceived, that is out of the fifty runs, how many times were the best individuals found on the deceptive area of the landscape. This gives us some idea of how the collaboration methods are handling the deceptiveness of the landscapes. Figures 7 and 10 show the deception rates over the fifty runs for two important statistics. The first, Population Trend, tells us if the final generation's best individual was on the deceptive area of the landscape. This tells us whether or not the population as a whole was led to a deceptive area. The second, Best, tells us if the best of the entire run was found in the deceptive area of the landscape. This will give us an idea if the algorithm as a whole was deceived, and if the Population Trend was an anomaly of chance.

Looking at figure 7, we can see some intriguing patterns in the deception rates and that differing collaboration methods do handle deceptiveness differently on the Deceptive One Ridge landscape. The first thing to note is that as soon as elitism is added to the collaboration method, that both the Best and Population Trend rates are correlated, almost always having equal values, while using purely random collaborators leads to a lower Best deception rate with a higher Population Trend rate. Further, the One Best collaboration method has a deception rate of 50% for both the statistics. This means that half the time it is both led to the deceptive side of the landscape and that it finds the best of the entire run on the deceptive side. However, other collaboration methods achieve deception rates as low as half that value, tending toward the deceptive area only a quarter of the time. Also, up to a point, increasing the number of collaborators gives use a better fitness. Despite the similar best of run values for both the One Ridge with no deception and the Deceptive One Ridge, deceptiveness certainly has an effect on it.

Now on figure 10, we can see why the One Best collaboration method performed much worse on the deceptive Two Ridge landscape. Again, the One Best method has a 50% deception rate, leading it to the deceptive optimal half of the time. Interestingly, the average best of run value on the Deceptive Two Ridge for the One Best collaboration method is the average of the two optima in the landscape. This time, however, there are much different patterns in the deception rate behavior for the different collaboration methods. Outside of the purely random collaboration methods, increasing the number of collaborators seems to enhance the capability to handle the deceptiveness on this landscape.

The ability of the collaboration method to bias our search can be seen on these four landscapes. Each landscape has its own best performer in relation to best of run fitness, or deception. If we know that our landscape has certain properties, then we can choose specific collaboration methods that perform well on it, while sacrificing capability for other landscapes. On the other hand, if we know nothing about the landscape we may need to use a collaboration method which has lower performance on specific landscape, but is robust against multiple landscapes. This follows from [15], with deceptiveness adding another variable to our set of problems, we must be careful to take it into account when biasing our search since robustness to deception may mean reduced performance on specific landscapes.

5. Conclusions and Future Work

The basic understanding of the behavior of co-evolutionary algorithms is just beginning. This paper examined a very narrow subset of the abilities of these algorithms and the parameters that can be modified. We have shown that various collaboration methods have properties that make them superior to others on specific landscapes. With the correct collaboration method, both the One Ridge and Two Ridge landscapes can have solutions found efficiently. We have also shown that by varying collaboration methods, we are able to overcome deceptiveness on landscapes. Further, collaboration methods that may be very good on one landscape, may not perform well across multiple landscapes, therefore a sub-optimal but more robust collaboration method may be preferable if the shape of the landscape is unknown.

This is only a small number of the possibilities for varying the abilities of the CCEA. Interactions between the various parameters that can be adjusted will provide an even greater range of capability. The experiments in this paper were limited to populations of size ten. This limits the number of paths that can be explored with the collaboration methods with an increased number of collaborators. Future work should look at the capability of these same collaboration methods with increasing population size. As population size increases, an increased number of collaborators may also be examined. We also limited the evaluations to a simple optimistic fitness assignment when there were multiple collaborators; however there are other ways to assign the fitness when there are multiple collaborators [12]. Understanding the interaction effects of these different parameters is a cornerstone to being able to effectively use co-evolutionary algorithms. Once these interactions are known, we can bias our algorithms in predictable ways to allow for better solutions to specific landscapes or generalize it for robustness in unknown landscapes.

6. REFERENCES

[1] K. De Jong. *Evolutionary Computation: A Unified Approach*. MIT Press, 2006.

[2] S. Ficici and J. Pollack. Challenges in Coevolutionary Learning: Arms-race Dynamics, Open-endedness, and Mediocre Stable States. In C. Adami, R. Belew, H. Kitano and C. Taylor, editors, *Proceedings of the Sixth International Conference on Artificial Life,* pages 238-247, MIT Press, 1998.

[3] D. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. New York, Addison-Wesley, 1998.

[4] D. Goldberg. *The Design of Innovation: Lessons from and for Competent Genetic Algorithms*. Boston, Kluwer, 2002.

[5] J. Holland. *Adaption in Natural and Aritificial Systems.* Ann Arbor, MI, University of Michigan Press, 1975.

[6] E. Popovici and K. De Jong. A Dynamical Systems Analysis of Collaboration Methods in Cooperative Co-evolution. In *AAAI Fall Symposium on coevolutionary and Coadaptive Systems*. AAAI Press, 2005.

[7] E. Popovici and K. De Jong, The Effects of Interaction Frequency on the Optimization Performance of Cooperative Coevolution, In *Proceedings of the 8th annual conference on Genetic and Evolutionary Ccomputation*. Seattle, Washington, 2006.

[8] E. Popovici and K. De Jong. Understanding Competitive Co-Evolutionary Dynamics via Fitness Landscapes. In S. Luke, editor, *AAAI Fall Symposium. Artificial Multiagent Learning.* AAAI Press, 2004.

[9] E. Popovici and K. De Jong. Understanding Cooperative Co-evolutionary Dynamics via Simple Fitness Landscapes. In *Proceedings of the 2005 conference on Genetic and evolutionary computation, GECCO-2005*, 2005.

[10] K. Potter and K. De Jong. A Cooperative Coevolutionary Approach to Function Optimization. In *Proceedings of the Third Conference on Parallel Problem Solving from Nature*, pages 249-257, Jerusalem, Israel, 1994. Springer.

[11] W. Spears. *Using Neural Networks and Genetic Algorithms as Heuristics for NP-complete problems.* Master's thesis, George Mason University, Fairfax, VA.

[12] R. P. Wiegand. *An Analysis of Cooperative Colevolutionary Algorithms.* PhD thesis, George Mason University, Fairfax, VA, 2004.

[13] R.P. Wiegand, W. Liles, and K. De Jong. An Emprical Analysis of Collaboration Methods in Cooperative Coevolutionary Algorithms. In L. Spector, editor, *Proceedings of GECCO 2001*, pages 1235-1242. Morgan Kaufmann, 2001.

[14] R.P Wiegand and M.A. Potter. Robustness in Cooperative Coevolution. In Proceedings of the 8th annual conference on Genetic and Evolutionary Computation. Seattle, Washington, 2006.

[15] D.H. Wolpert and and W.G. Macready. No Free Lunch Theorems for Optimization. In *IEEE Transactions on Evolutionary Computation*. Volume 1, Issue 1, 1997.

Using Genetic Algorithms to Improve the Performance of Logistic Regression Models

Anthony Vaiciulis
University of Central Florida
Orlando FL 32816
anthony.vaiciulis@gmail.com

ABSTRACT
Logistic regression models are often used in data mining problems in which an accurate, interpretable predictive model is required. Some disadvantages of these models include the fact that the user must choose which predictors enter the model, what functional form these predictors have, how to incorporate nominal-scale predictors into the model, which interaction terms enter the model, and how to handle observations with missing values. For all but the smallest datasets, the large search space defined by these choices prevents an exhaustive search to find the optimal set of choices. This paper addresses the first three of these concerns, describing how to create accurate, interpretable predictive models using logistic regression tuned by a genetic algorithm. The inclusion of interaction terms would be a natural extension of the work presented here.

General Terms
Algorithms, Performance.

Keywords
Logistic Regression, Predictive Modeling, Genetic Algorithm, Evolutionary Computation.

1. INTRODUCTION
Data mining is a means of extracting previously unknown, actionable information from large amounts of data using sophisticated, automated algorithms to discover hidden patterns, correlations and relationships [2,3,8,12,16,18,22,23,29,31,38]. Building predictive models is often an important part of the data mining process. In a typical case, historical data is used to create a mathematical model which is then applied to new data to make predictions. This is an example of a supervised learning method, in which a training dataset consists of a response variable plus the predictor variables. Note that in data mining applications, a higher emphasis is typically placed on predictive accuracy rather than on ability of the model to yield insights into the nature of the relationships among the predictors and between the predictors and the response. Logistic regression is often used in predictive modeling when the response is binary because this type of model is often easily interpreted and can be very accurate. With other models such as artificial neural networks, a potential for higher predictive accuracy comes at the expense of reduced interpretability -- the magnitude and direction of the relationship between each predictor and the response may not be clear, for example.

Some disadvantages of logistic regression models include the fact that the user must choose which predictors enter the model, what functional form these predictors have, how to incorporate nominal-scale predictors into the model, which interaction terms enter the model, and how to handle observations with missing values. For all but the smallest datasets, the large search space defined by these choices prevents an exhaustive search to find the optimal set of choices. In this paper we describe the use of a genetic algorithm to tune the parameters (choices) of a logistic regression model. This technique efficiently searches the large space, balancing the conflicting forces of high predictive accuracy and low complexity as defined by the number of predictors in the model.

Section 2 presents background information on logistic regression and previous work done on this using evolutionary algorithms. Section 3 describes some details of the genetic algorithm including how the fitness function is defined. In Section 4 we describe the datasets used to test the performance of the technique. Section 5 describes the experiments and results, incrementally increasing the sophistication of the model at each stage. In Section 6 we state the conclusions and offer ideas for future work.

2. BACKGROUND
2.1 The Logistic Regression Model
To understand the logistic regression model and introduce some terminology we take an example of measuring the age of a tree using several characteristics of the tree. A dataset may consist of the height, girth, species and age of 100 trees. A specific analysis may use the height and girth as interval-scale (numeric) predictors, species as a nominal-scale (categorical) predictor, and age as an interval-scale response. So this dataset has 100 observations, 3 predictors and an interval-scale response, age. Here, a simple linear regression model may be appropriate:

$$\text{age} = \beta_0 + \beta_1(\text{height}) + \beta_2(\text{girth}) + \beta_3(\text{species})$$

where the β_is are coefficients to be determined by a least squares fitting process.

Logistic regression models [21] are used in problems in which the response is binary. In the tree example above, we may simply be interested in whether or not the tree is more than five years old. We define a new, binary response variable "mature" with value 1 for a tree with age >= 5 and value 0 for a tree with age < 5. A logistic regression model may be formed by defining p as the probability that the response takes the value 1, P(mature=1):

$$\log(p/(1-p)) = \beta_0 + \beta_1(\text{height}) + \beta_2(\text{girth}) + \beta_3(\text{species})$$

Given enough training data, the values of the β_i coefficients can be estimated by iterative numerical optimization techniques, which results in a completely specified logistic regression model that can be used to predict the value of 'p' for any observation containing the predictors height, girth and species.

The simple model above is most likely not the optimal model. It may be that introducing a transformation, such as height^2, will produce a model that fits the data better. It may be that the inclusion of a multiplicative interaction term such as height*girth will improve the model performance. If the nominal-scale predictor species has more than two distinct values, it cannot be used as shown in the model above, with a single coefficient. It must be converted into a numeric form that the logistic regression model understands. Thus the logistic regression model demands much of the user and various strategies have been created to develop such models [19].

One reward of a well-developed model is an easy interpretation of the β_i coefficients. For example, let us assume that the logistic regression model shown above fits the data well, height is measured in meters, and the value of the β_1 coefficient is found to be +0.28. The interpretation is that the estimated odds that the tree is at least 5 years old (i.e. mature) increases by a factor $e^{0.28} = 1.3$ for every additional meter of height, adjusting for the girth and tree species.

Given a dataset with many predictors, it is usually desirable to select that subset of the predictors which yields a model that fits the data well. This results in a more compact representation of the data. There are several well known methods of choosing the subset of predictors in a regression problem including "forward", "backward" and "stepwise" methods [21]. These methods do not necessarily give the best subset and they contain somewhat subtle flaws in the statistical assumptions they make [14,19]. An exhaustive search of all possible combinations of predictors and transformations takes a prohibitively long time.

2.2 Previous Work

For several reasons, an evolutionary algorithm (EA) approach to determining the form of a logistic regression model is quite natural as a regression model satisfies many of the rules of thumb suggesting when an EA approach may be successful [28]. First, the search space is very large due to the many predictors, the many possible transformations of continuous variables, and the various ways of collapsing levels of categorical variables. An exhaustive search of all possible combinations of options is not feasible. Second, the search space is not well understood and may have many local optima. There is no reason to believe a gradient approach will do well. Third, the fitness function is noisy. The fitness of a regression model is evaluated on a test sample of observations and has some random error. Methods such as cross-validation can be used to smooth out these errors by evaluating fitness of a solution on many samples, but still there will be uncertainty. EAs can perform robustly in the presence of small amounts of noise. Finally, finding a good, but not necessarily the best possible regression model in terms of predictive accuracy, is acceptable.

Some research has already been done on how to use EAs to solve regression problems. Koza used genetic programming techniques to perform "symbolic regression" -- finding the functional form as well as the coefficients in fitting a curve to data points [25]. Siedlecki and Sklansky [34] studied the use of genetic algorithms (GAs) [11,17,28] for the general problem of predictor selection. Some studies have used a GA to select predictors in logistic regression [36,37]. They did not consider transformations of predictors, which potentially can yield a more accurate model. They also did not consider nominal-scale predictors, which occur often in real-world datasets. Krause and Tutz used a GA for predictor selection and for finding the best transformations in a generalized additive model, which is less interpretable than a standard logistic regression model because there is no parametric form for the predictor transformations [26,27]. They also did not consider nominal-scale predictors. Broadhurst et al. use GAs only as a method for variable selection in a linear regression problem. [5].

Bala et al. use GAs to create a predictive model but the underlying learning machine optimized by the GA is a decision tree rather than a logistic regression model. This is an interesting approach as a decision tree implicitly handles transformations of predictors, interactions among predictors, nominal-scale predictors and predictors with missing values. Thus the GA is used only for selection of predictors. A single predictor tree, however, has several disadvantages including instability in predictions, relatively poor predictive accuracy, and difficulty in interpretation for trees of larger size. Combinations of trees eliminate some of these disadvantages at the cost of a great loss of interpretability.

3. THE GENETIC ALGORITHM

In this paper we describe a technique of using a GA to optimize a logistic regression model. The GA is used to determine the subset of interval-scale predictors in each model, the functional form of each of these predictors, and the subset of nominal-scale predictors in the model. We use a conventional bit-encoded GA. Unless otherwise noted, the following GA parameters are used: population size of 50, 50-70 generations, bit-flip mutation with rate 0.01, two-point crossover with rate 0.7, best fitness individual appears unchanged in next generation with probability one ("elitism"), tournament selection of size n=4 and probability 0.9 to select the most fit individual among the four. Most results are presented as averages over multiple runs, each with a different random number seed. Standard deviations calculated from the values of multiple runs. More details on the bit encoding of the GA are given in Section 5.

Each individual in the population represents the choices made regarding which predictors to include and how to transform them. Given these choices the fitness can be evaluated. The fitness has two components: predictive accuracy and complexity. Here, we define complexity as the fraction of total predictors which are used in the model. For example, if a dataset contains a total of 100 predictors and a specific model includes only 25 of them, the complexity of that model is 0.25. This complexity is a measure of the number of degrees of freedom of a specific model.

Several methods of evaluating predictive accuracy were considered. Bootstrap methods [7,13] were discarded in favor of a five-fold cross-validation [20,30,32] method. In both cases, area under the ROC curve (AUC) is used as the measure of predictive accuracy [20,35]. The ROC curve is a plot of true positive rate versus false positive rate, with a larger area under the curve indicating a model with higher predictive accuracy. In statistical terms, AUC is equal to the value of the Wilcoxon-Mann-Whitney test statistic and is also the probability that the classifier will score higher on a randomly drawn positive sample than on a randomly drawn negative sample.

In this study, the fitness function to be maximized is defined as a linear combination of AUC and complexity:

$$\text{fitness} = \text{AUC} - \alpha(\text{complexity})$$

where AUC ranges from 0.5 (random guessing) to 1.0 (perfect predictions), complexity ranges from 0 (no predictors are used in model) to 1 (all predictors are used in model), and α is set to 0.1 unless otherwise noted. Larger values of AUC and smaller values of complexity are desirable. The R software environment is used for all facets of analysis presented in this paper unless otherwise noted.

4. DESCRIPTION OF DATASETS

Several datasets were used to evaluate the results of the GA-tuned logistic regression model. Each dataset was required to satisfy the following conditions: 1) the response should be binary so logistic regression is a suitable candidate model, 2) the number of observations should be at least several hundred to minimize problems that arise when developing a predictive model for small datasets, 3) the number of predictors should be relatively large (e.g. > 10) to provide a problem in which an exhaustive search is truly prohibitive, and 4) the amount of preprocessing already performed on the dataset should be minimized. Requirement 4 is included because, for example, some datasets have already had nominal-scale predictors transformed into numeric predictors by an unknown or possibly sub-optimal method. Some details

4.1 M2007 Dataset

This dataset, a subset of the dataset used in the M2007 data mining competition, consists of 10,669 observations, a binary response and 166 predictors of which 38 are interval-scale and the rest binary. The response variable identifies those people who are likely to be high-revenue customers for a magazine company. The binary predictors are used in the model directly with no transformations. We use this dataset with many predictors in the first phase of algorithm development, when the GA is used only to select subsets of predictors. Five runs are performed and results are used to study some details of how the GA is operating and how its performance compares with forward, backward and stepwise selection methods.

4.2 Email Spam Dataset

This dataset is used extensively in Hastie et al. [20], where enough information is given to allow direct comparisons of performance between the GA-tuned logistic regression model and the models in Hastie. The spam dataset contains 4601 observations, a binary response, and 57 interval-scale predictors. The response indicates whether or not an email is spam (unsolicited junk mail). We use this dataset in the second phase of algorithm development in which transformations of predictors are allowed.

4.3 Graduate School Enrollment Dataset

This dataset consists of 2665 observations, a binary response and 35 predictors of which 24 are interval-scale and 8 are nominal-scale. The binary response indicates whether or not a student admitted to a graduate program at the University of Central Florida (UCF) chooses to enroll. We use this dataset in the third phase of algorithm development in which nominal-scale predictors are included.

5. EXPERIMENTS AND RESULTS

5.1 Interval-scale Predictors Only

We first used the M2007 dataset to study the performance of the GA compared to several standard methods of predictor subset selection. Each individual in the population consists of 166 bits, one for each predictor in the dataset. It was found, relative to the GA parameters stated in Section 3, that a smaller population, a smaller number of generations, less or no elitism, and larger population initialization probabilities generally yielded models with lower fitness. Figure 1 shows the fitness of the best model in each generation with vertical bars indicating the standard deviation. After 50 generations, the GA clearly finds a logistic regression model which has a fitness at least equivalent to the best fitness reached by the forward, backward and stepwise predictor selection methods indicated by the three horizontal lines. The forward, backward and stepwise methods were implemented in SAS.

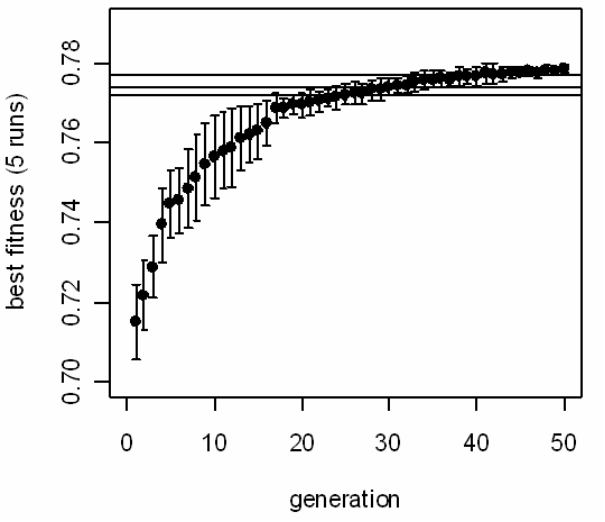


Figure 1. Fitness vs. generation for M2007 dataset.

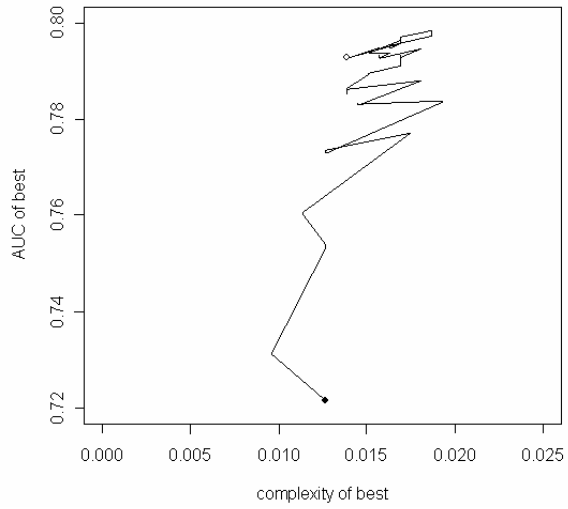


Figure 2. AUC vs. complexity for run 1 on M2007 dataset.

Figure 2 shows the predictive accuracy of the model as measured by AUC vs. the complexity of the model for the highest fitness model in each generation in run 1. The solid circle represents the 1st generation while the empty circle represents the 50th generation. The trade-off between predictive accuracy and complexity as codified in the fitness function manifests itself as a zigzag line in this plot.

Figure 3 shows the AUC of the highest fitness model for GAs (plus signs) versus the number of predictors in the model. The solid circles are for forward, backward and stepwise methods. Generally speaking, models closer to the top left corner of this plot are better. The five runs of the GA produce relatively fit models, which tend to have higher AUC and/or lower complexity than the models resulting from forward, backward or stepwise methods.

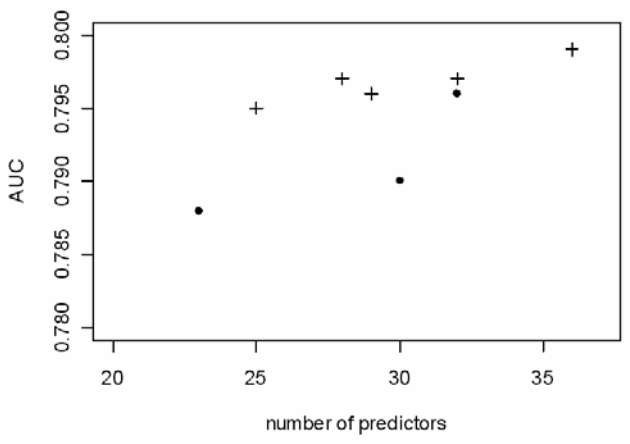


Figure 3. AUC vs. number of predictors for M2007 dataset.

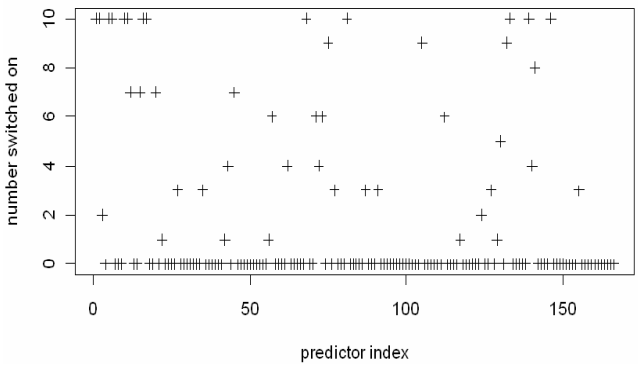


Figure 4. Number of predictors in best model vs. predictor index in run 1 on M2007 dataset.

Figure 4 shows that even in the last ten generations of one of the runs the number of predictors in the highest fitness model is not constant. Although each of these best models contains 25-30 predictors, only 13 of the predictors are included in all ten best models. Thus there is a collection of different models which have similarly high fitness.

For the first 20 of the 166 predictors, Figure 5 shows which of them are included (indicated by an X) in the highest fitness model of each GA run and in the forward, backward and stepwise models. Several predictors such as 1 and 2 seem to be very useful as all methods include these predictors. Other predictors such as 11 and 20 are found to be useful only by the GA-tuned logistic regression model. This table also provides clear evidence that the best models found by the GA are not identical for the different runs.

Index	Step	Backward	Forward	Run1	Run2	Run3	Run4	Run5
1	X	X	X	X	X	X	X	X
2	X	X	X	X	X	X	X	X
3	X	X	X			X	X	
4								X
5	X	X	X	X	X	X	X	X
6	X	X	X	X	X	X	X	X
7	X	X	X					X
8					X			
9	X	X	X			X		
10	X	X	X	X	X	X	X	X
11				X			X	
12	X	X	X	X	X	X	X	
13			X		X	X	X	
14	X	X	X		X			
15	X	X	X	X	X	X	X	X
16	X	X	X	X	X	X	X	X
17	X	X	X	X	X		X	X
18	X	X	X					
20				X	X			

Figure 5. A comparison of which predictors are included in various models for the M2007 dataset.

5.2 Addition of Transformations

The GA was then modified to allow for transformations of the interval-scale predictors. Figure 6 shows, from top to bottom, the following twelve transformations allowed for each predictor x: x^2, $x^{1.75}$, $x^{1.5}$, $x^{1.25}$, x^1, log(x), $x^{0.5}$, $x^{0.33}$, $x^{-0.33}$, $x^{-0.5}$, x^{-1}, x^{-2}. These transformations are encoded by four bits for each predictor. If the number of these four bits with value 1 is zero or one, then there is no transformation (x^1). This choice, plus the relative sparseness of 1s in the initial population, biases the initial search to predictors with no transformation. Each predictor is now encoded by five bits – one for the ON/OFF switch and four for the transformation. The spam dataset was used to study the GA performance with this modification. Each individual now consists of 57*5=285 bits.

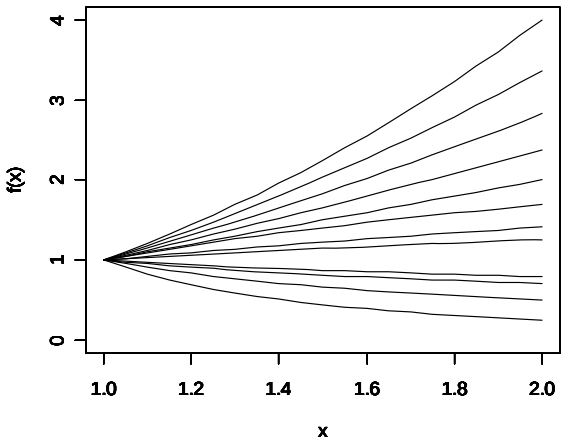


Figure 6. The 12 transformations allowed for each predictor.

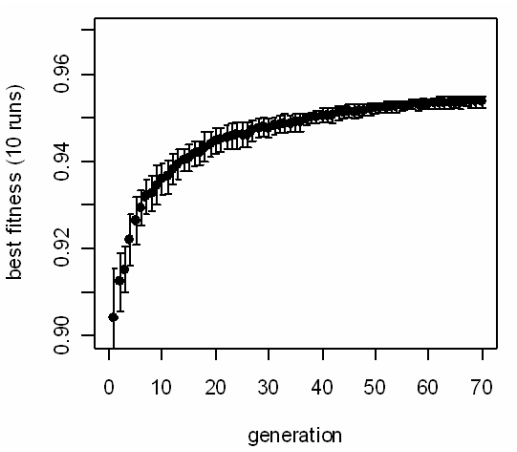


Figure 7. Fitness vs. generation for spam dataset.

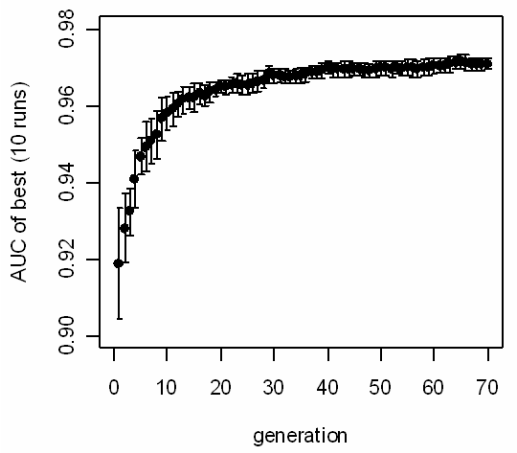


Figure 8. AUC vs. generation for spam dataset.

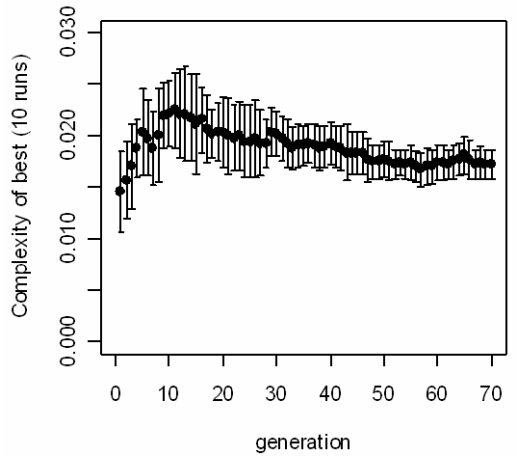


Figure 9. Complexity vs. generation for spam dataset.

Figures 7, 8 and 9 show the best fitness in each generation as well as the tow components of fitness, AUC and complexity. In the later generations, a combination of small increases in AUC and decreases in complexity drive the fitness higher. As a reference point for the complexity plot, note that a model using 10 of the 57 available predictors has a complexity of (10/57)*0.1 = 0.0175.

The highest fitness model in each run is next trained on the same data used to train by Hastie et al.[20] Then this model is used to predict the 1536 observations in the test set, as defined by Hastie, to compare directly with Hastie's results. A model in run 5 with only 9 predictors had the lowest misclassification rate of 6.6% in the test set. This can be compared to the GAM model in Hastie which has 16 predictors and a misclassification rate of 5.3%. Note that although the GA-tuned logistic regression model has a larger misclassification rate, it is a simpler model because it has only about half as many terms and its predictors have a well-specified functional form. In the GAM model of Hastie, the term for each predictor is the result of a scatter plot smooth which does not give a parametric form. Other models developed by Hastie for this dataset include MART (4.0% error, 48 predictors), MARS (5.5% error, 60 independent basis functions), and CART (8.7% error, 17 predictors). The MART, MARS and CART models all include interactions between predictors making them more difficult to interpret. The 6.6% error and 9 predictors of the GA-tuned logistic regression model is a much more compact representation of the dataset but its predictive accuracy is not as great as some of Hastie's models. Note that changing the definition of the fitness function, specifically decreasing the α parameter, would result in a more accurate but more complex model.

5.3 Addition of Nominal-Scale Predictors

Finally, the graduate enrollment dataset is used to study how well the GA performs. A smoothed weight-of-evidence method is used to convert the multiple, distinct levels of each nominal-scale predictor into a single numeric predictor [15,33]. The initial population probability was increased from 0.1 to 0.2 to achieve a higher AUC since slightly more complicated models result. The α parameter in the fitness function was decreased from 0.1 to 0.05 also to produce models with a higher AUC. The specific goal for this dataset was to find a model with an AUC value greater than or equal to 0.784, but with a complexity lower than that of the model currently being used for this dataset by the UCF Division of Graduate Studies. The current model has an AUC of 0.784 with 19 terms. Nine of the terms are main effects (single predictors) and 10 of the terms are multiplicative interactions of two predictors. Due to the many interactions, the model is more complicated than a model with 19 main effect terms. The GA-tuned logistic models have AUC of about 0.784 or 0.785 on average (see Figure 10). More importantly they have only 11 predictors and all are main effect terms. This is much less complicated than a model with 19 terms with interactions.

Figures 11 and 12 show the fitness and complexity of the best model in each generation. The initial fitness is driven by an increase in complexity which leads to much higher predictive accuracies, but this trend is reversed in the later generations when the GA is able to find higher fitness models with lower complexities. Figure 13 shows the Hamming mean value for all unique pairs of individuals in the population. It clearly shows that the diversity of the population increases initially as the GA explores the search space and then decreases in later generations.

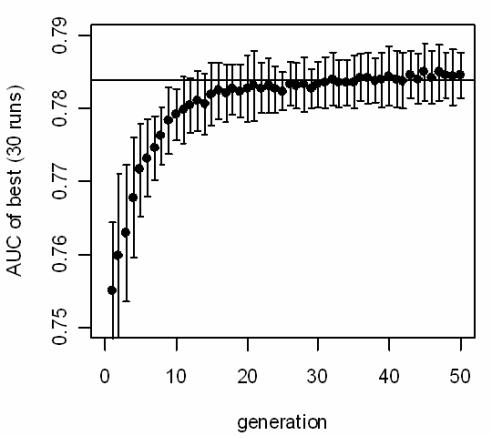


Figure 10. AUC of best model vs. generation for graduate enrollment dataset.

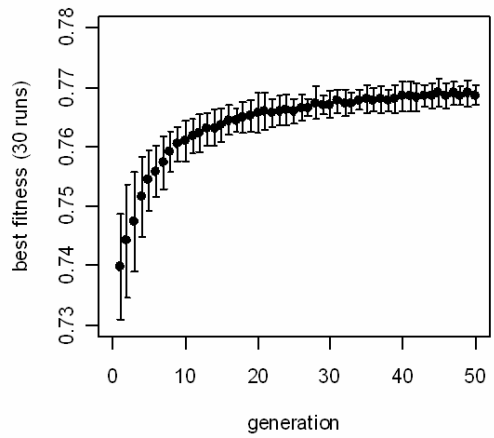


Figure 11. Best fitness vs. generation for graduate enrollment dataset.

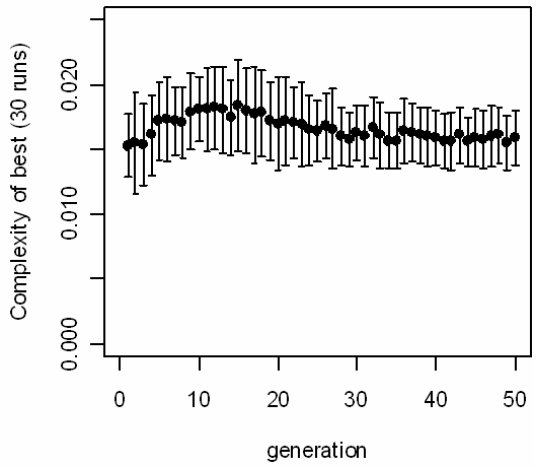


Figure 12. Complexity of best model vs. generation for graduate enrollment dataset.

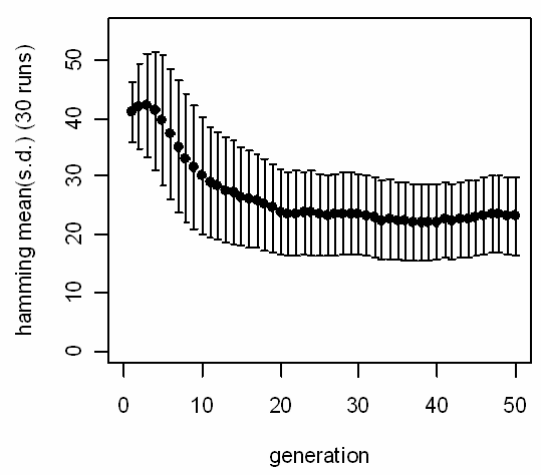


Figure 13. Hamming mean vs. generation for graduate enrollment dataset.

6. CONCLUSIONS

Using several different datasets, we have demonstrated the use of a genetic algorithm to tune a logistic regression model which handles transformations of interval-scale predictors as well as automatic incorporation of nominal-scale predictors. The technique can be used to find a logistic regression model of high accuracy and relatively low complexity.

The work presented here can be extended in several ways. Terms representing multiplicative interactions between predictors can be allowed in the regression model. This may increase fitness at the cost of reduced interpretability of the model. The smoothing parameter for nominal-scale predictors can be allowed to vary. On a broader scale, the use of multiobjective genetic algorithms [6,9,10] can be investigated. In this paper, we have used a weighted linear combination of two conflicting objectives to produce a single objective function. With a multiobjective GA, however, the conflicting objectives remain separate and the algorithm yields a collection of best fitness models with various combinations of values for the different objectives. Some work has already been done with multiobjective GAs in predictive modeling [4,24] as there is often a trade-off between, for example, between raw performance and complexity of a predictive model.

The technique described in this paper still has the disadvantage of not incorporating missing values automatically. Various kinds of selection, mutation, and crossover were not explored. Each fitted model contains information about how significant each predictor is based on the estimates of the coefficients and their standard errors. Perhaps this information can be fed back to the GA to improve performance. There are many avenues for future investigation.

7. ACKNOWLEDGMENTS

The second author wishes to thank the University of Central Florida department of Statistics and Actuarial Science for making SAS software available.

8. REFERENCES

[1] Bala, J., DeJong, K. Huang, J. Vafaie, H. and Wechsler, H. Hybrid Learning Using Genetic Algorithms and Decision Trees for Pattern Classification. *Proceedings of the International Joint Conferences on Artificial Intelligence (IJCAI)*, Montreal, Canada, August 19-25, 1995.

[2] Berry, M.J.A., Linoff, G. *Data Mining Techniques*. Wiley, New York, NY, 1997.

[3] Berry, M.J.A., Linoff, G. *Mastering Data Mining*. Wiley, New York, NY, 2000.

[4] Bhattacharyya, S. Evolutionary Algorithms in Data Mining: Multi-Objective Performance Modeling for Direct Marketing. *Proceedings of KDD 2000*, Boston, MA, 2000.

[5] Broadhurst, D., Goodacre, R. Jones, A., Rowland, J. and Kell, D. Genetic algorithms as a method for variable selection in multiple linear regression and partial least squares regression, with applications to pyrolysis mass spectrometry. *Analytica Chimica Acta* 348:71-86, 1997.

[6] Coello Coello, C.A., Van Veldhuizen, D.A. and Lamont, G.B. *Evolutionary Algorithms for Solving Multi-Objective Problems*. Kluwer Academic Publishers, New York, NY, 2002.

[7] Davidson, A.C. and Hinkley, D.V. *Bootstrap Methods and their Application*. Cambridge University Press, New York, NY, 1997.

[8] Dasu, T. Johnson, T. *Exploratory Data Mining and Data Cleaning*. Wiley-Interscience, New York, NY, 2003.

[9] Deb, Kalyanmoy. *Multi-Objective Optimization using Evolutionary Algorithms*. Wiley, Chichester, England, 2001.

[10] Deb, K., Pratap, A., Agarwal, S. and Meyarivan, T. A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation* 6(2), 2002.

[11] DeJong, Kenneth A. *Evolutionary Computation: A Unified Approach*. MIT Press, Cambridge, MA, 2006.

[12] Duda, R.O., Hart, P.E., Stork, D.G. *Pattern Classification*. Wiley-Interscience, New York, NY, 2001.

[13] Efron, B. and Tibshirani, R. *An Introduction to the Bootstrap*. Chapman & Hall/CRC, Boca Raton, FL, 1993.

[14] Faraway, Julian J. *Practical Regression and Anova using R*. http://cran.r-project.org/doc/contrib/Faraway-PRA.pdf, 2002.

[15] Georges, J. Using non-numeric data in parametric prediction. *Proceedings of Seventh Annual Data Mining Conference (M2004)*, Las Vegas, Nevada, 2004.

[16] Giudici, Paolo. *Applied Data Mining*. Wiley, New York, NY, 2003.

[17] Goldberg, D. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, Reading, MA, 1989.

[18] Hand, D. Mannila, H., Smyth, P. *Principles of Data Mining*. MIT Press, Cambridge, MA, 2001.

[19] Harrell, Frank E. *Regression Modeling Strategies*. Springer Science+Business Media. New York, NY, 2001.

[20] Hastie, T., Tibshirani, R., and Friedman, J. *The Elements of Statistical Learning*. Springer-Verlag, New York, NY, 2001.

[21] Hosmer, David W. and Lemeshow, Stanley. *Applied Logistic Regression*. Wiley-Interscience, New York, NY, 2000.

[22] Kantardzic, Mehmed. *Data Mining: Concepts, Models, Methods and Algorithms*. Wiley-Interscience, New York, NY, 2003.

[23] Kantardzic, M.M., Zurada, J., Editors, *Next Generation of Data-Mining Applications*. Wiley-Interscience, New York, NY, 2005.

[24] Kim, Y., Street, W.N. and Menczer, F. An Evolutionary Multi-Objective Local Selection Algorithm for Customer Targeting. *Proceedings of Congress on Evolutionary Computation (CEC-01)*, Seoul, Korea, 2001.

[25] Koza, J.R. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA, 1992.

[26] Krause, Rudiger. *Genetic Algorithms as Tool for Statistical Analysis of High-Dimensional Data Structures*. Ph.D. Thesis, Ludwig-Maximilians University, Munich, Germany, 2004.

[27] Krause, R. and Tuts, G. Genetic Algorithms for the Selection of Smoothing Parameters in Additive Models. *Computational Statistics* 21(1):9-31, 2006.

[28] Mitchell, Melanie. *An Introduction to Genetic Algorithms*. MIT Press, Cambridge, MA, 1996.

[29] Mitchell, T.M. *Machine Learning*. McGraw-Hill, Boston, MA, 1997.

[30] Picard, R. and Cook, D. Cross-Validation of Regression Models. *Journal of the American Statistical Association*, 79(387):575-583, 1984.

[31] Pyle, Dorian. *Data Preparation for Data Mining*. Morgan KaufmannSan Francisco, CA, 1999.

[32] Shao, J. Linear Model Selection by Cross-Validation. *Journal of the American Statistical Association* 88(422):486-494, 1993.

[33] SAS Press, *Advanced Predictive Modeling Using SAS Enterprise Miner 5.1*.

[34] Siedlecki, W. and Sklansky, J. A note on genetic algorithms for large-scale feature selection. *Pattern Recognition Letters* 10:335-347, 1989.

[35] Sing, T., Sander, O., Beerenwinkel, N., Lengauer, T. ROCR: visualizing classifier performance in R. *Bioinformatics* 21 (20):3940-3941, 2005.

[36] Stacey, A. and Kildea, D. Genetic Algorithm Search for Large Logistic Regression Models with Significant Variables. *Proceedings of the 22nd Int. Conf. on Information Technology Interfaces (ITI)*, Pula, Croatia, June 13-16, 2000.

[37] Vinterbo, S. and Ohno-Machado, L. A genetic algorithm to select variables in logistic regression: example in the domain of myocardial infarction. *Journal of the American Medical Informatics Association*, 6(Suppl.):984-988, 1999.

[38] Webb, Andrew. *Statistical Pattern Recognition*. Wiley, New York, NY, 2002.

Reproducing an Evolution of Artificial Plants

Tommy McDaniel
University of Central Florida
4000 Central Florida Blvd.
Orlando, FL 32816
tommy@cs.ucf.edu

ABSTRACT

In this paper, a reproduction of experiments from Marc Toussaint's paper *Demonstrating the Evolution of Complex Genetic Representations: An Evolution of Artificial Plants* [11] will be described.

1. INTRODUCTION

The author's original intention was to do a project involving robotics. However, after consultation with Dr. Garibay, it was decided that such a project would be too risky for the scope of this semester project, since it had two parts that could go wrong: the evolutionary part, and the neural network part. A couple of ideas were presented to the author: one was to do an updated version of a survey that was published a few years ago, and the other was to implement Toussaint's paper. The latter option was chosen.

1.1 Overview of the Paper

The paper that was implemented describes a method of evolving artificial plants. The plants are represented using a form of L-systems, as proposed by Prusinkiewicz and Hanan [10]. These consist of structures similar to context-free grammars. A plant consists of an embryo Ψ and a set Π of operators $\langle \pi_1, \pi_2, \ldots, \pi_n \rangle$. The embryo consists of a string of symbols from the alphabet $\Sigma = \{A, B, C, \ldots, P\}$. The operators π_i consist of a promoter, which is a single element of Σ, and a string of symbols from Σ, which was never given a clear name in the paper[1].

Starting from the embryo Ψ, one creates Ψ_1 by taking each operator π_i in order and replacing each instance of its promoter in the embryo with its genes. One creates Ψ_2 in the same manner, but applying the operators to Ψ_1 this time. This process is continued until an arbitrary stopping point is reached, at which point the final Ψ_n represents the plant's phenotype.

[1]This author referred to these strings as *genes* in his implementation of the paper.

Computing plant fitness requires simulating a three-dimensional version of the plant. Given an overhead view of the plant, every green pixel[2] adds to the fitness an amount dictated by the height at that pixel. This is counterbalanced by a negative weight, calculated based on how big the plant is.[3]

The concept of generative representations is certainly not unique to this paper. Hornby and Pollack have published much research on generative representations in recent years [1, 2, 3, 4, 5, 6, 7, 8]. Kicinger, Arciszewski, and De Jong recently compared the use of parameterized representations versus generative representations to design skyscrapers [9]. The main idea of Toussaint's paper was what it called 2^{nd}-*type mutations*. These were mutations of the operators that were designed to be neutral (i.e., that resulted in the same phenotype).

2. PROCEDURE

Toussaint stated in his paper that source code was available on his website. However, no such source code was to be found, and the author contacted Toussaint requesting a copy of the code. After a communications problem, Toussaint was never heard from again. It was at that point that the author began implementing the entire paper on his own.

The author chose C as his implementation language. Toussaint said in his paper that "Evolving such plant structures already gets close to the limits of today's computers, both, with respect to memory and computation time." Therefore, performance was of the essence. The graphical parts were developed with the OpenGL family of toolkits. Sampling from Poisson distributions was done with the free GNU Scientific Library (GSL).[4] Images of the plants were created with the free GD library.[5] All development was done in Linux.

The first step was to be able to simulate arbitrary plants, without regard for finding their fitness or evolving their representations. This simulator portion was successfully created and tested. Obtaining overhead information about the plants simply required placing the camera at the proper position over the plant and reading OpenGL's green and depth buffers.

[2]Since leaves and only leaves are green, although, in a simulation, the choice of green is arbitrary.
[3]This is literally an estimate for how much the plant weighs.
[4]`http://www.gnu.org/software/gsl`
[5]`http://www.libgd.org`

The evolutionary part, however, did not go nearly as smoothly. The main problem was that C does not have built-in string data structures, so one was written by hand and expanded upon as the need arose. A string data structure was an obvious need, since the lion's share of dealing with these representations involved string manipulation, due to the fact that we are basically dealing with context-free grammars. However, rich string data structures are *extremely* prone to off-by-one errors. This part of the program took the vast majority of the development effort.

2.1 Experimental Parameters

The experiments had many parameters that could affect the outcome. The values used are documented in Table 1. Most values came from Toussaint's original experiments. The main difference is the value of ϱ, which was found by trial and error; values that were too small quickly resulted in enormous, disorganized plants that led to expected simulation times measured in months for a single run, while values that were too large resulted in plants being unable to evolve. The values of α and β were fixed, as in Toussaint's second trial, due to the fact that otherwise they tended to go as close to zero as possible. Plants were initialized with the same genotype as in Toussaint's experiments.

All runs were of 1,000 generations. The only known divergence from Toussaint's implementation is in how a plant's weight is calculated. Toussaint used a recursive function to calculate the weight, whereas this implementation simply counts the number of phenotypic elements (branches and leaves).

3. RESULTS

After initial testing, seven quality runs were done with the final settings. The statistics of the best plant of each run are in Table 2, and aggregate statistics of these best plants are in Table 3. An interesting phenomenon that occurred in many cases[6] was that the best plants were produced well before the end of the run, usually shortly after fitness began to increase significantly from near zero, followed by a decline for the rest of the run.

There is an interesting correlation between embryo size and fitness, as shown in Table 4. If we ignore run 3, then for the other six runs, increasing embryo sizes result in lower fitnesses. Since a small embryo must of necessity rely heavily on its operators to create a large plant, whereas a large embryo can be closer to a direct encoding, this appears to support the utility of generative representations.

3.1 Run 1

The average fitness, best fitness, and standard deviation of the fitness per generation are plotted in Figure 1. The best plant is shown in Figure 2, and its operators are listed in Table 5. Of the seven plants listed in this paper, this one had the second-largest embryo and phenotype, but the least operators and second-smallest weight. Its identifying characteristic is its apparent thickness, which is caused by its second operator, which produces 10 leaves rotated about a single point. This effect is greatly magnified by its first operator.

[6]Both in these results and in unpublished results.

3.2 Run 2

The average fitness, best fitness, and standard deviation of the fitness per generation are plotted in Figure 3. The best plant is shown in Figure 4, and its operators are listed in Table 6. This plant had by far the largest embryo, and the second-lowest fitness, but was otherwise unremarkable. The looping structures are caused by its first operator, which produces four leaves in a row, followed by a single rotation of δ degrees.

3.3 Run 3

The average fitness, best fitness, and standard deviation of the fitness per generation are plotted in Figure 5. The best plant is shown in Figure 6, and its operators are listed in Table 7. This plant had the smallest phenotype size, lowest weight, and lowest fitness. It is notable that this small, "premature" plant emerged in the second-fastest time of all the runs. It is distinguished by the polygonal structures that its operators create.

3.4 Run 4

The average fitness, best fitness, and standard deviation of the fitness per generation are plotted in Figure 7. The best plant is shown in Figure 8, and its operators are listed in Table 8. This plant emerged very quickly, nearly 300 generations faster than the best plant of any other run, and had the second-highest weight. The degree to which fitness fell into a continual decline for the rest of the run after its emergence is conspicuous. Similarly to run 2, the hoops that form the plant are caused by the interaction of the three operators, which combine to form a structure consisting of many sides with four leaves each followed by a single turn of δ degrees.

Parameter	Value	Description
δ	20	Rotation angle
b	20	Length of each side of plant bounding cube
T	1	Iterations through the list of operators
ϱ	3×10^{-7}	Plant weight multiplier
α	0.01	First-type mutation rate
β	0.3	Second-type mutation rate
μ	30	Parent population size
λ	100	Offspring population size
M_{max}	1,000,000	Phenotype size limit
R_{max}	100	Maximum operators per plant
U_{max}	40	Size cutoff for symbol duplication mutations

Table 1: Experiment Parameter Values

Run	1	2	3	4	5	6	7
Generation	728	852	684	398	960	959	981
Embryo size	10,858	42,722	2,141	5,830	248	1,129	2,704
Phenotype size	442,549	166,950	86,362	208,299	156,310	203,856	835,998
Number of operators	2	3	5	3	8	6	8
Weight	76,440	145,229	70,044	180,499	146,596	189,695	160,542
Fitness	0.26016	0.248408	0.218884	0.391968	0.469252	0.447236	0.44273

Table 2: Statistics of Best Plant per Run

	Average	Standard Deviation
Generation	794.6	210.7
Embryo size	9,376	15,137.8
Phenotype size	300,046.3	261,169.5
Number of operators	5	2.45
Weight	138,435	47,467.8
Fitness	0.354091	0.107628

Table 3: Aggregate Statistics of Best Plants per Run

Run	5	6	7	4	1	2
Embryo size	248	1,129	2,704	5,830	10,858	42,722
Fitness	0.469252	0.447236	0.44273	0.391968	0.26016	0.248408

Table 4: Correlation Between Embryo Size and Fitness

Length of Genes	Operator
3	D → AAA
64	A → ADDDDDDDDDDDDDDDDDIEEIEEEEIEEIEEDDDDDDDDIEEKEEMEEEEILEEIEEIEEIIE

Table 5: Operators of Best Plant of Run 1

Length of Genes	Operator
9	A → IAIAIAIAB
1	C → F
2	F → IA

Table 6: Operators of Best Plant of Run 2

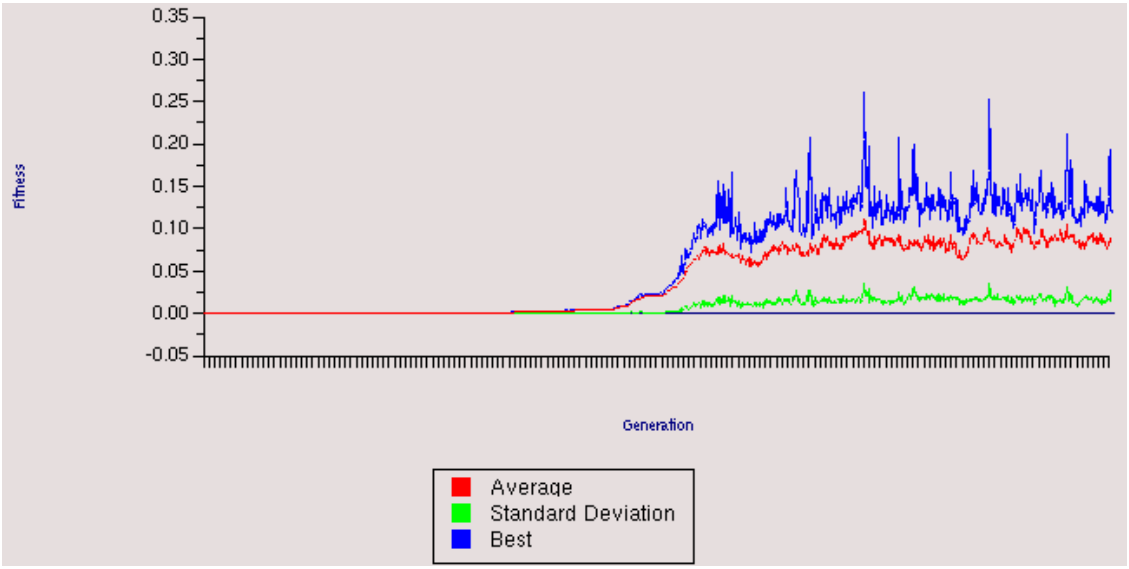
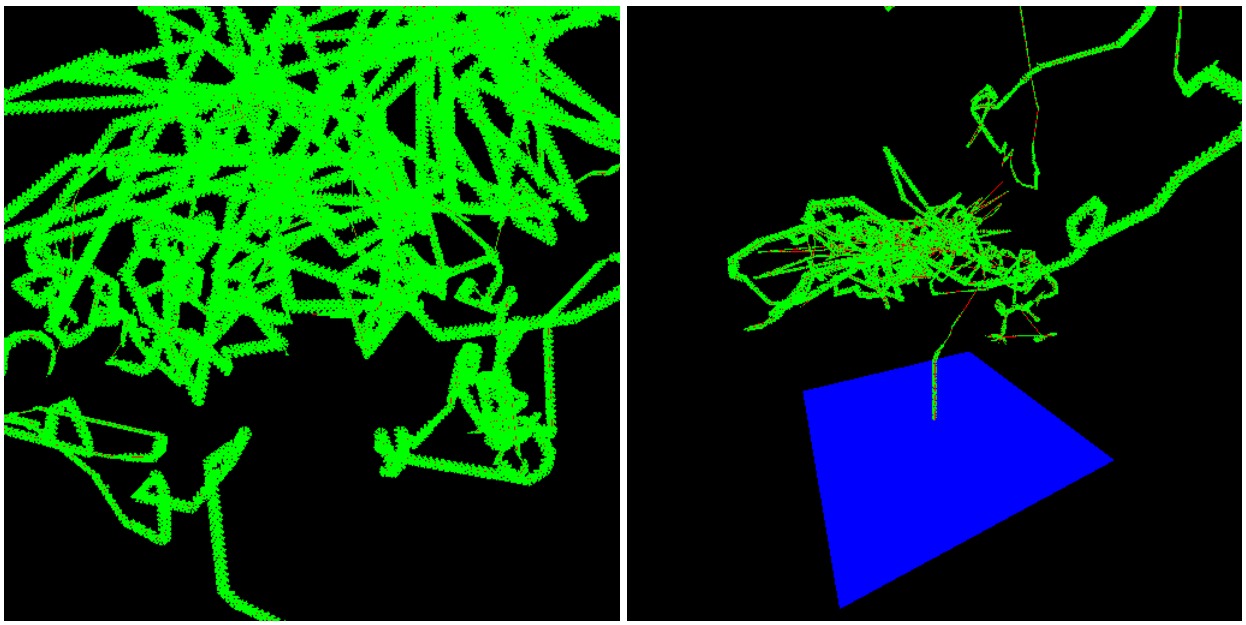


Figure 1: Average fitness, best fitness, and standard deviation of fitness in run 1



(a) Overhead view (b) Glamour view

Figure 2: Best plant of run 1, generation 728, fitness = 0.26016

Length of Genes	Operator
7	A → DOOFFDD
35	F → OEIIMAEEOPJELAEIIPIPIFIHEEGEEEEEEEE
2	P → NA
2	E → IA
1	N → I

Table 7: Operators of Best Plant of Run 3

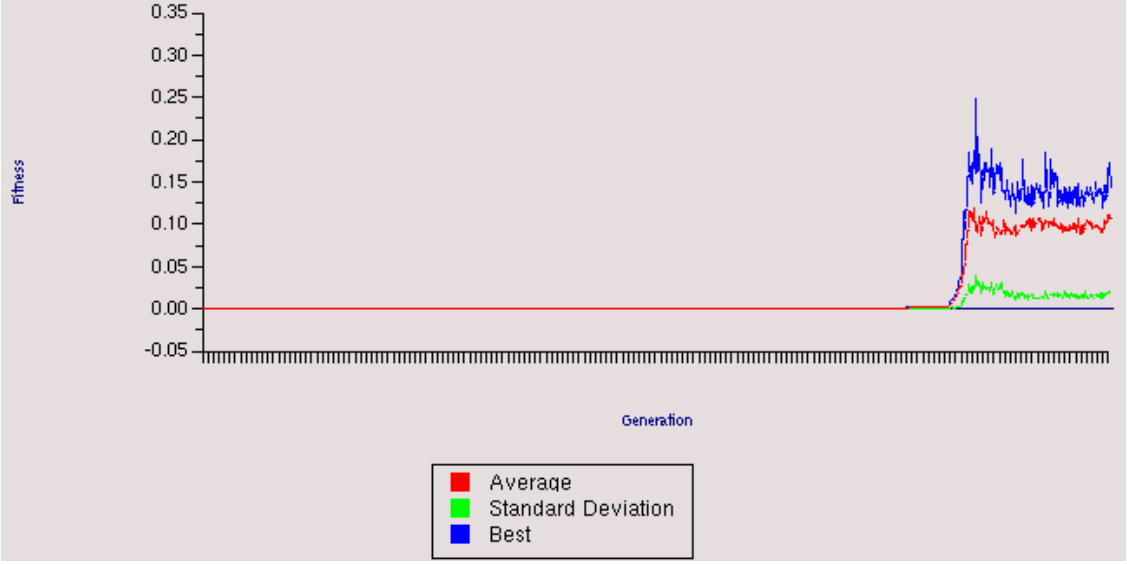
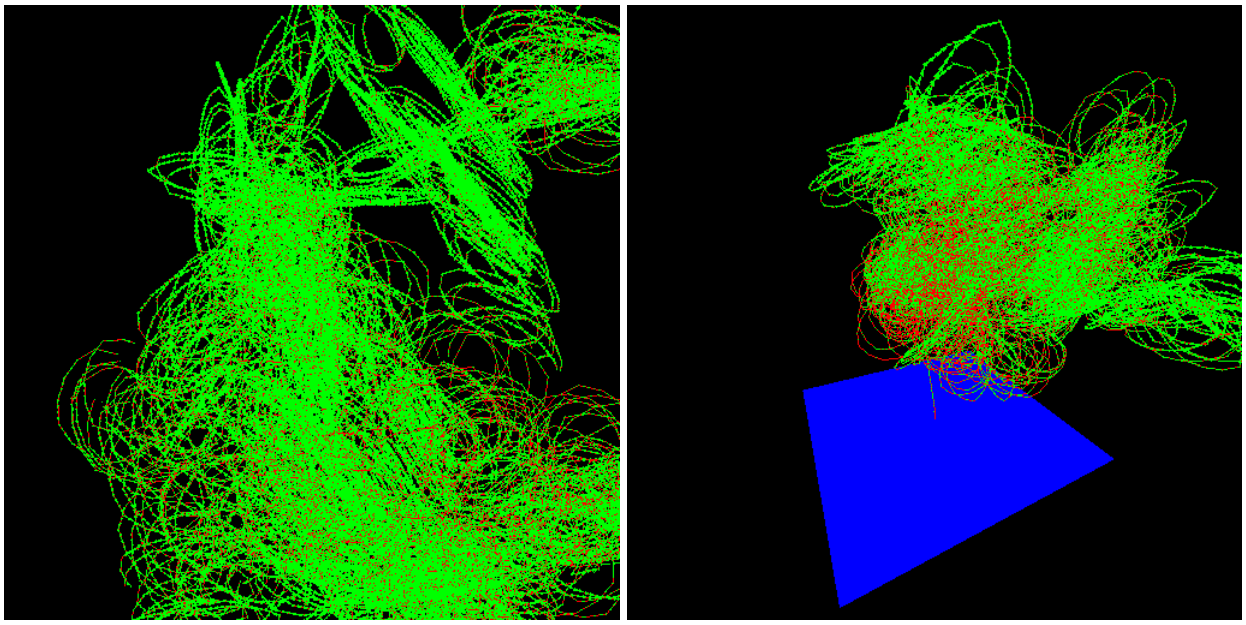


Figure 3: Average fitness, best fitness, and standard deviation of fitness in run 2



(a) Overhead view (b) Glamour view

Figure 4: Best plant of run 2, generation 852, fitness = 0.248408

Length of Genes	Operator
7	F → FFFFFFE
7	F → IDAIAMM
2	M → IA

Table 8: Operators of Best Plant of Run 4

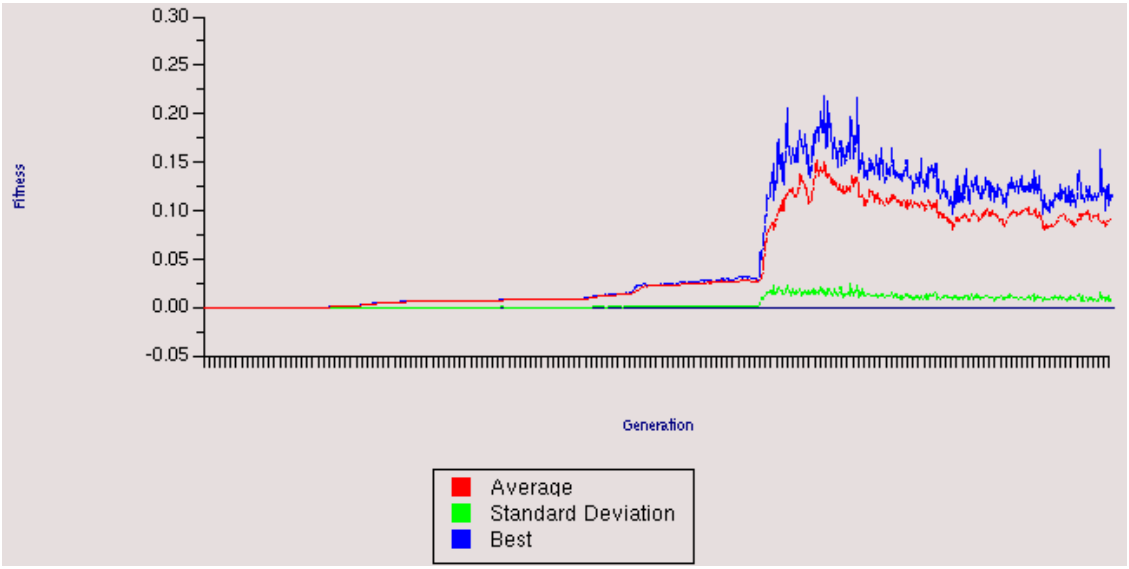
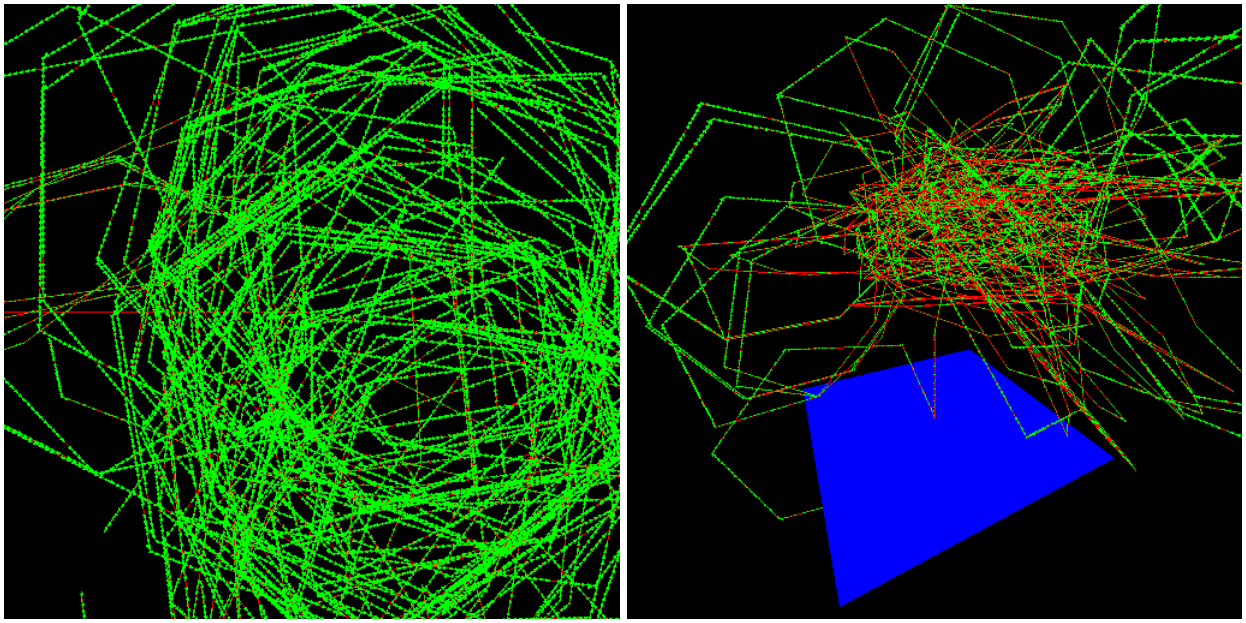


Figure 5: Average fitness, best fitness, and standard deviation of fitness in run 3



(a) Overhead view (b) Glamour view

Figure 6: Best plant of run 3, generation 684, fitness = 0.218884

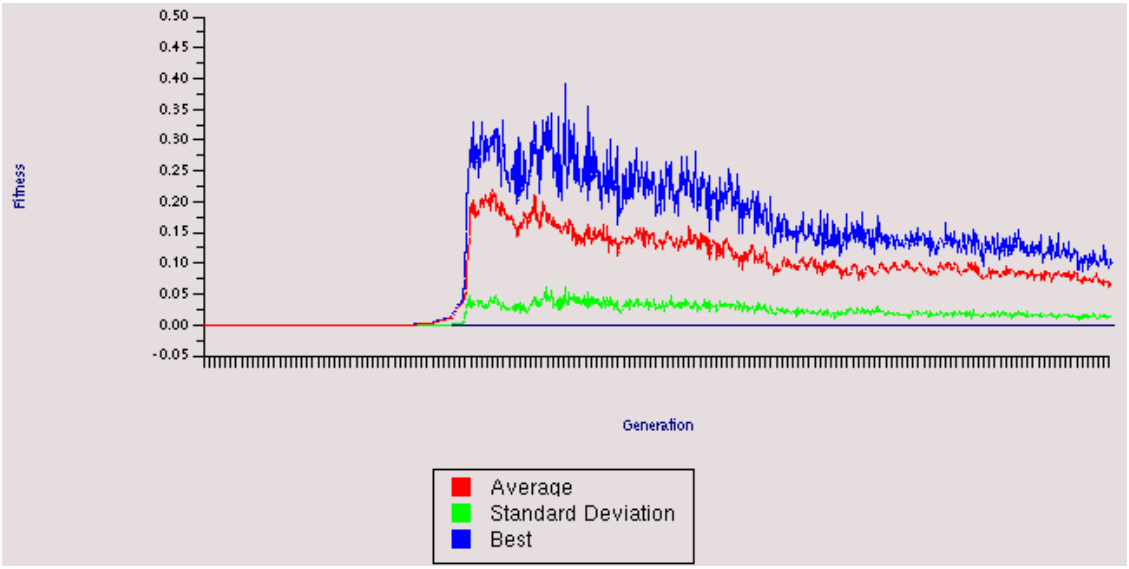
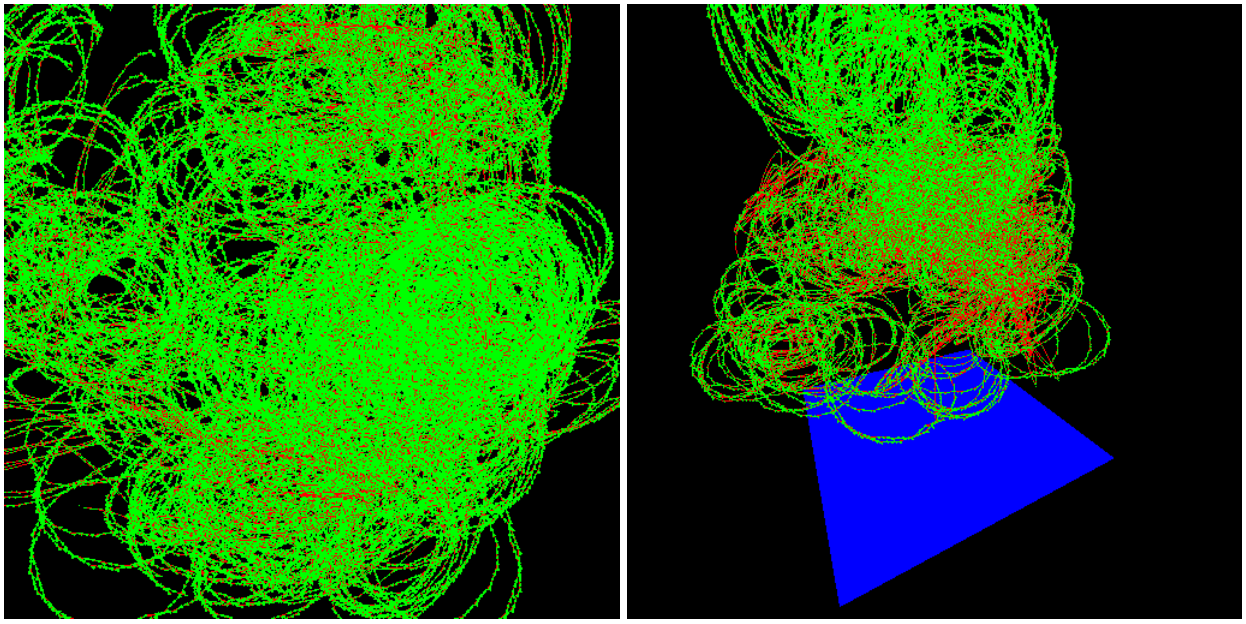


Figure 7: Average fitness, best fitness, and standard deviation of fitness in run 4



(a) Overhead view (b) Glamour view

Figure 8: Best plant of run 4, generation 398, fitness = 0.391968

3.5 Run 5

The average fitness, best fitness, and standard deviation of the fitness per generation are plotted in Figure 9. The best plant is shown in Figure 10, and its operators are listed in Table 9. This plant had the smallest embryo, the second-smallest phenotype, the highest fitness, and tied for the most operators. This was the first run where fitness increased nearly constantly after it got going, with its best plant appearing at the second-latest time of any of the runs.

3.6 Run 6

The average fitness, best fitness, and standard deviation of the fitness per generation are plotted in Figure 11. The best plant is shown in Figure 12, and its operators are listed in Table 10. This plant had the second-smallest embryo, the highest weight, and the second-highest fitness. It has a markedly polygonal structure, with the polygons offset slightly from each other. This run was marked by a prolonged period of increasing fitness, followed by somewhat of a decline, and ending with a final burst that produced better plants than had been produced before the decline. This was the only run whose best plant came during an upswing following a meaningful period of decline.

3.7 Run 7

The average fitness, best fitness, and standard deviation of the fitness per generation are plotted in Figure 13. The best plant is shown in Figure 14, and its operators are listed in Table 11. This plant emerged the latest of any of the plants in this paper, had by far the biggest phenotype, and tied for the most operators. It has a wide, ribbon-like structure.[7]

4. CONCLUSIONS

The experiments from Toussaint's paper can be considered to have been successfully duplicated. While C or something similarly fast may have been the wise choice for implementing this paper, it would have probably been much better to use preexisting string data structures instead of creating new ones from scratch. In hindsight, this project had the same problem as a robotics project: multiple points of failure. A robotics project could have had problems with either the evolutionary part or the neural network part, but this project also had two points of failure, the evolutionary part and the graphical part.

Source code and data can be found at the author's website at `http://cs.ucf.edu/~tommy`.

5. REFERENCES

[1] G. S. Hornby. Generative representations for evolving families of designs. In *2003 Genetic and Evolutionary Computation Conference (GECCO 2003)*, 2003.

[2] G. S. Hornby. Functional scalability through generative representations: the evolution of table designs. *Environment and Planning B: Planning and Design*, 31(4):569–587, July 2004.

[3] G. S. Hornby. Measuring, enabling and comparing modularity, regularity and hierarchy in evolutionary design. In *2005 Genetic and Evolutionary Computation Conference (GECCO 2005)*, 2005.

[4] G. S. Hornby, H. Lipson, and J. B. Pollack. Evolution of generative design systems for modular physical robots. In *IEEE International Conference on Robotics and Automation*, 2001.

[5] G. S. Hornby, H. Lipson, and J. B. Pollack. Generative representations for the automated design of modular physical robots. *IEEE Transactions on Robotics and Automation*, 19(4):703–719, 2003.

[6] G. S. Hornby and J. B. Pollack. The advantages of generative grammatical encodings for physical design. In *Congress on Evolutionary Computation*, 2001.

[7] G. S. Hornby and J. B. Pollack. Evolving l-systems to generate virtual creatures. *Computers and Graphics*, 25(6):1041–1048, 2001.

[8] G. S. Hornby and J. B. Pollack. Creating high-level components with a generative representation for body-brain evolution. *Artificial Life*, 8(3), 2002.

[9] R. Kicinger, T. Arciszewski, and K. D. Jong. Parameterized versus generative representations in structural design: An empirical comparison. In *2005 Genetic and Evolutionary Computation Conference (GECCO 2005)*, 2005.

[10] P. Prusinkiewicz and J. Hanan. *Lindenmayer Systems, Fractals, and Plants*. Springer, New York, 1989.

[11] M. Toussaint. Demonstrating the evolution of complex genetic representations: An evolution of artificial plants. In *2003 Genetic and Evolutionary Computation Conference (GECCO 2003)*, pages 86–97, 2003.

[7] The author is reminded of IDE cables.

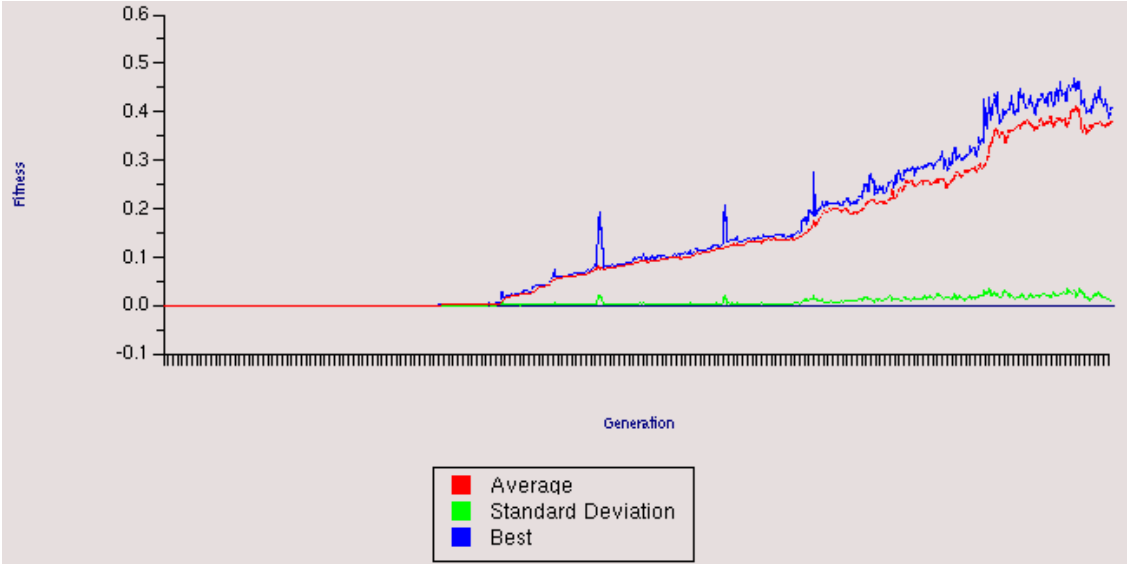
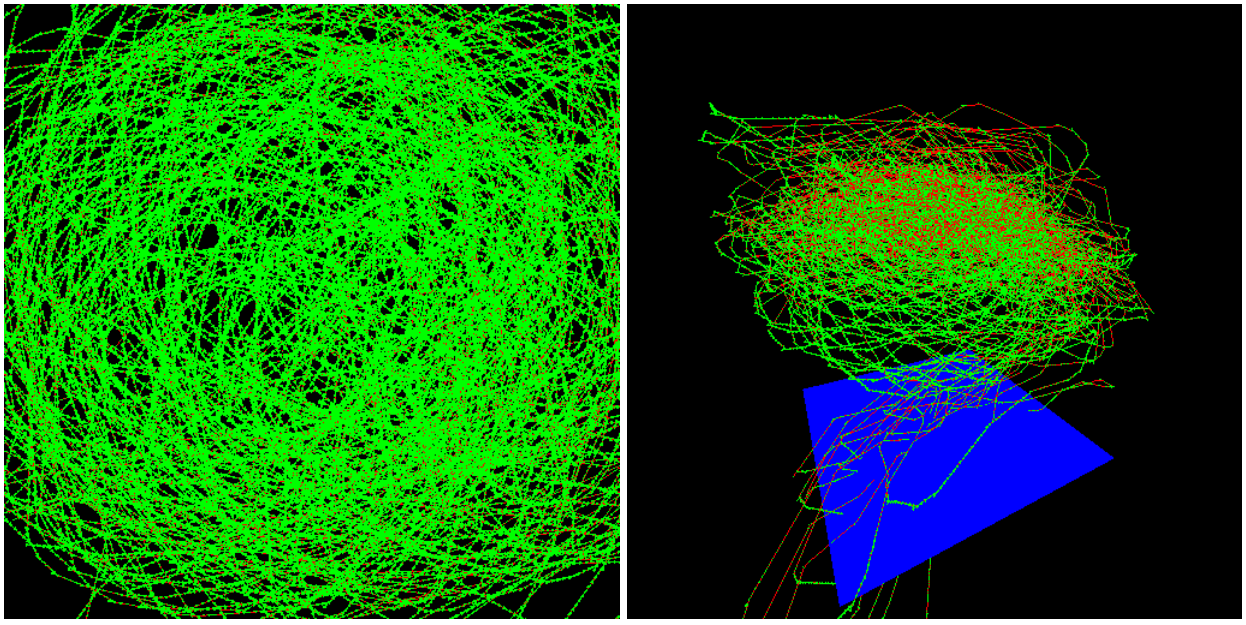


Figure 9: Average fitness, best fitness, and standard deviation of fitness in run 5



(a) Overhead view (b) Glamour view

Figure 10: Best plant of run 5, generation 960, fitness = 0.469252

Length of Genes	Operator
36	A → OOOOCAAIIAIJIAIHIHEDAGIIAPOAIMAOACOO
40	O → ICAAAIIDIIMIDDIIOMICPIPJMICIDAAAADABAAAC
5	A → IIIII
3	I → IAI
3	M → AIA
3	L → HOE
1	N → L
1	L → L

Table 9: Operators of Best Plant of Run 5

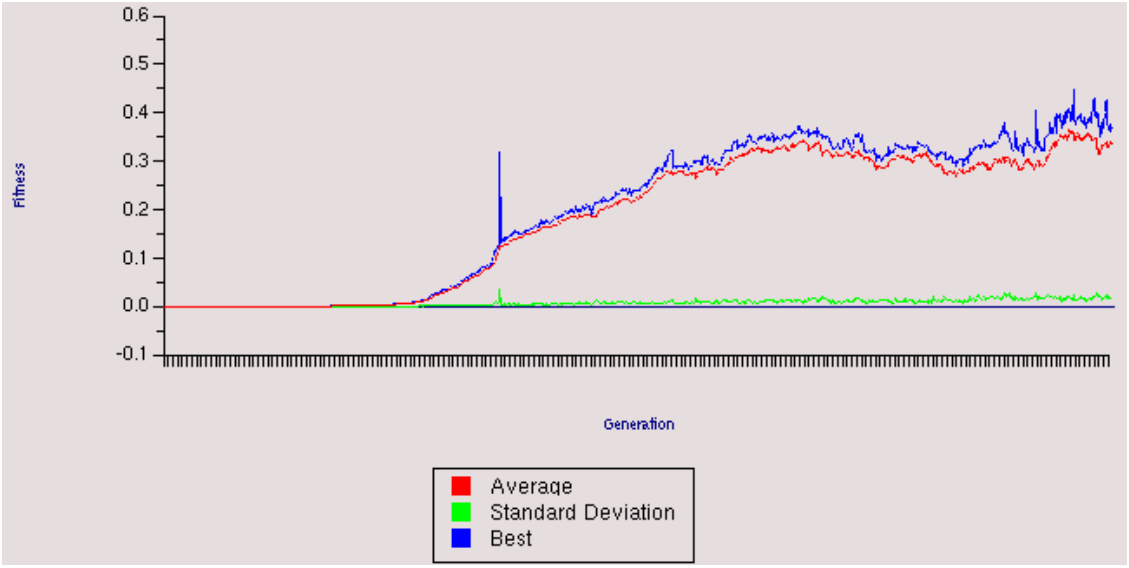
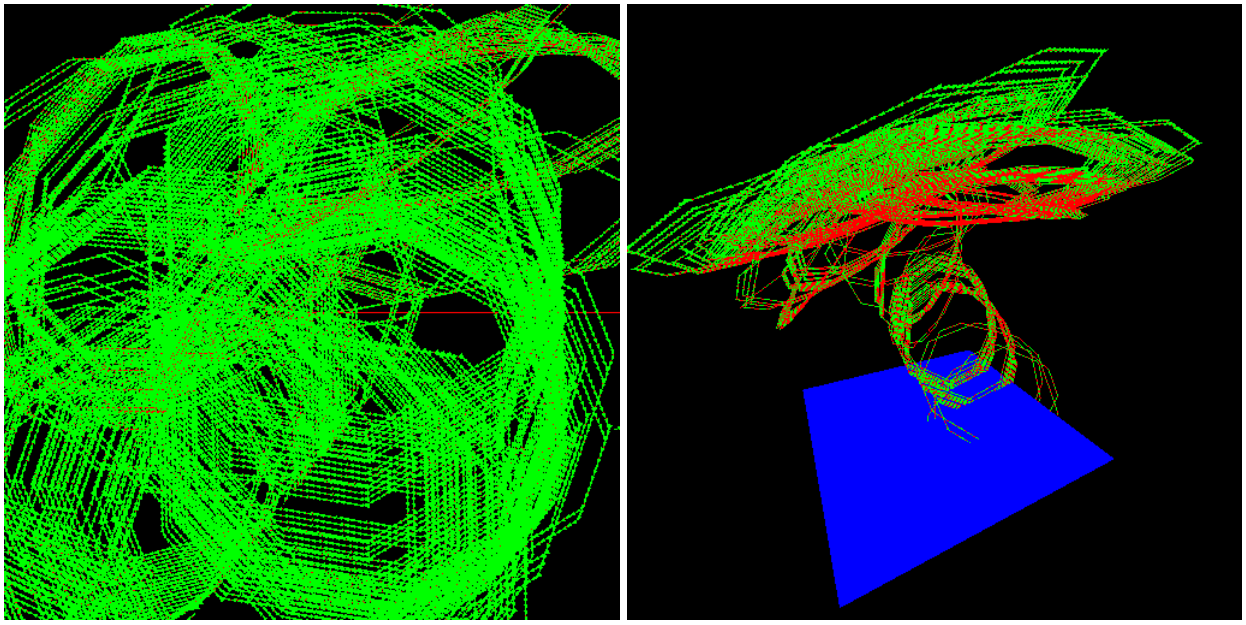


Figure 11: Average fitness, best fitness, and standard deviation of fitness in run 6



(a) Overhead view (b) Glamour view

Figure 12: Best plant of run 6, generation 959, fitness = 0.447236

Length of Genes	Operator
13	E → FFFFFFFFFIIIA
60	F → FEIAEIIAGIAIMIIKIIJIIFIAIIAIKEIAEIIAGIJNKHIIMIIFIIAIAIAFIAIA
2	F → FF
4	F → GGGG
0	L →
4	G → IIIA

Table 10: Operators of Best Plant of Run 6

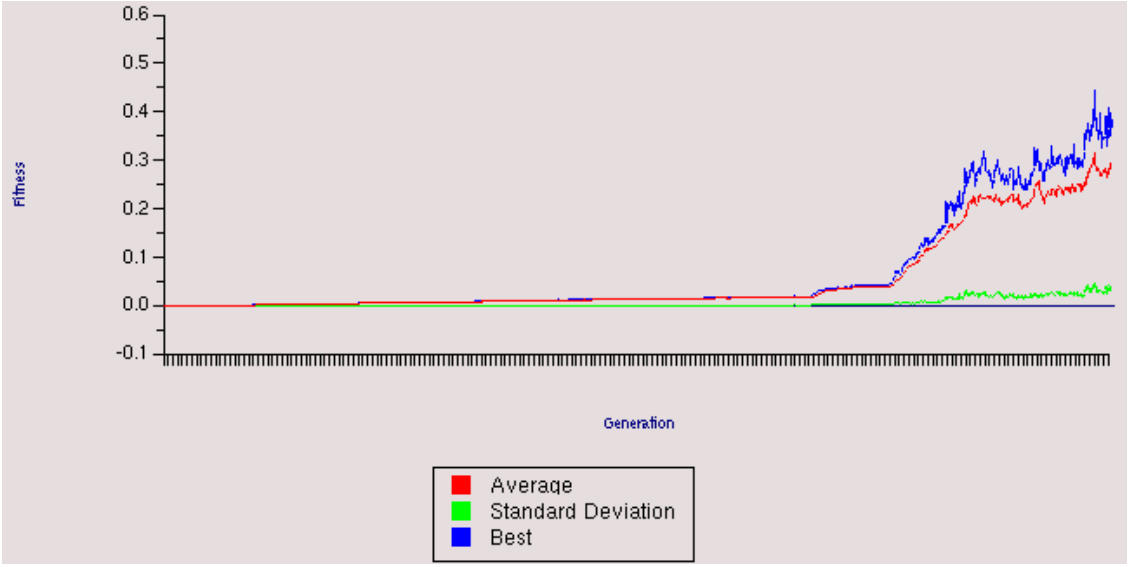
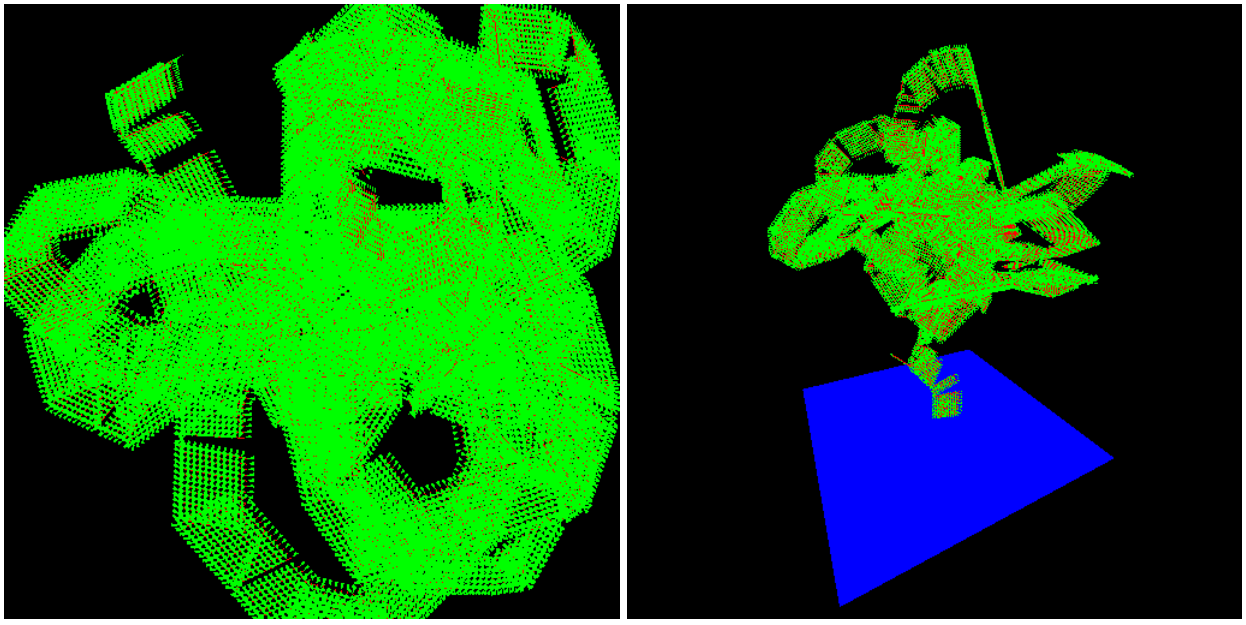


Figure 13: Average fitness, best fitness, and standard deviation of fitness in run 7



(a) Overhead view (b) Glamour view

Figure 14: Best plant of run 7, generation 981, fitness = 0.44273

Length of Genes	Operator
8	I → AAHNECLI
10	B → INIAAAIBIA
17	A → BBILMMMMMMMMMMMEA
12	M → BBIHKCGBEGEA
2	A → OA
5	B → HEEEE
2	O → BI
2	B → II

Table 11: Operators of Best Plant of Run 7

Evolving A Simple Instinctive Behavior

Victor C. Hung
School of Electrical Engineering and Computer Science
University of Central Florida
Orlando, Florida
victor@isl.ucf.edu

ABSTRACT

This paper investigates a primitive implementation of an instinct-based behavioral model in an artificial agent. It serves as an early attempt in devising an artificially intelligent entity whose decisions are based on instinct, as defined by field of ethology. The agent behavior is represented as a finite state machine. Three experiments were conducted to investigate two key features of the behavior model: 1) the external environment, and 2) the number of states needed to represent behavior. The resultant instinctive behavior set was based on a very simple representation structure whose evolutionary mutability is sensitive to the stability of its environment factors. In general, the primary goal of this work is to use a genetic algorithm to automatically produce a basic behavior representation based on instincts.

Categories and Subject Descriptors

I.2.11 [**Artificial Intelligence**]: (Distributed Artificial Intelligence)Multiagent systems

General Terms

Behavior Representation

Keywords

instincts, evolution, genetic algorithm

1. INTRODUCTION

Creating agent behavior representations is a time-consuming effort, often requiring meticulous attention to small details and countless hours of development and testing [1]. An automatically generated behavior model could replace these knowledge-based models, eliminating this extended development period. This paper investigates producing an agent behavior representation based on instincts without human intervention. The primary motivation of this endeavor is to devise an agent behavior that is driven by a set of primitive, innate behavioral mechanisms.

A specific application for an instinct-based behavioral model is the action-response systems of artificial agents in a simulated environment. These agents may exist as autonomous, interactive entities in training scenarios, such as those found in military exercises [9] or in video games [5]. In many cases, these agents are implemented with scripted behaviors, resulting in actions dictated by a simple input-output lookup table [12]. The work in this paper presents an extremely primitive behavior model of very basic instincts. The idea is to provide a method to automatically derive this instinctive behavior for use in more complex behavior representations, such as those found in the aforementioned applications.

In this paper, the evolution of innate behaviors in actual animals will be simulated using a genetic algorithm-based method in a population of artificial agents. These evolved behaviors will be viewed as *instincts*, as defined by ethology. Two particular issues will be examined pertaining to this evolution of intrinsic behaviors. The first will concern the data structure representation of the agent behavior. The second issue concerns the role of environment in the evolutionary process. The experiments in this work were designed to provide insight into these items.

The following sections give some related background information, followed by a detailed account of the work at hand, from the problem formulation to the experiment results. The paper will conclude with a wrap-up discussion, a brief mention of additional work to be done, and a conclusion section.

2. BACKGROUND

The impetus for this work remains the idea that an agent's behavioral response set could feasibly be generated using evolutionary means, without the need for an extended period of hand-modeled behavior development. In nature, instincts serve as an automatic behavior set for which animals base their primitive responses upon. According to the work of ethologists, instincts result from a process of behavioral evolution [2]. For artificial agents, evolutionary algorithms could also be applied to produce similar effects. With this in mind, a few assumptions will be made regarding agent behavior. The first assumption claims that *instinctive* behavior in animals is established as a priori constructs. This suggests that no pre-learned skills are needed to execute instinctive actions. Secondly, instincts are the result of evolution running its course upon the genetic makeup of animals. Thus, both environmental and natural factors have an in-

fluence on the instinctual characteristics of every species. This section provides background material regarding these assumptions.

2.1 Instincts

When discussing natural behavior, the idea of instincts immediately comes to mind. Natural instincts result from a process of behavioral evolution [6]. Over the course of millions of years, certain instincts may withstand the test of time in an animal species. They essentially serve as a priori behaviors instilled in an animal at birth, often directly related to a being's ability to survive to adulthood. Instincts differ from reflexes in that the former deals with an individual's ability to respond in a rationally motivated manner, while a reflex simply is an intrinsic, physiological impulse, usually triggered at a muscular, rather than cognitive, level [10]. It is also noted that instincts are widely regarded in modern psychology as general guidelines or motivations, rather than specific skills or actions.

2.2 Behavioral Evolution

Ethology describes the study of animal behavior from a biological, as opposed to psychological, point of view. Hence, ethology is a separate field from psychology, that offers its own stance on instinctive behavior. In general, ethologists examine instincts under an empirical, physiological light, while psychologists treat instincts with a more abstract, comparative approach.

As mentioned before, the development of instincts in animals is viewed as an evolutionary process - Lorenz established this standpoint as a staple of ethological studies [2]. Over long periods of time, multitudes of organisms are spawned with varying flavors of innate tendencies. As per the mechanics of evolution, only those organisms that are good enough to survive will be able to pass on their genetic make-up to the next population. Within this genetic coding lies the set of intrinsic instincts. Tinbergen asserts that these instincts exist as Fixed Action Patterns (FAPs) [14]. FAPs are those behaviors that are unlearned but are essential for an animals survival. They are triggered by innate releasing mechanisms. Examples of FAPs include bees mating dances, gulls egg-laying patterns, and minnows feeding behaviors.

2.3 Genetic Algorithms and Evolving Behaviors

With the advent of evolutionary algorithms in computer science, it is easily envisioned how such solutions may be applied to this problem of devising instinctive behaviors for artificial entities. Stanley, Bryant, and Miikkulainen [13], and Portegys [11] have approached this idea of performing genetic algorithms to produce innate behavior sets. Stanley et al. [13] present the idea of evolving agents to improve their survival fitness in a video game environment. The skill set of these agents is primarily of a military nature. Portegys [11] evolves an agent that is skilled at the *Monkey and Bananas* problem.

While both of these efforts promote the evolution of behaviors, their evolved behaviors are often characterized as activities that require a high level of skill. The purpose of this paper is to present a method that operates at a very basic skill level - behaviors often described as instincts. The works of Eck [3] and Inoue and Kobayashi [8] provide good starting points in this particular realm. Both of these papers use a simple artificial life model that deals with basic predator-prey dynamics in a grid-based world. This paper serves to build upon the foundations set forth by these contributions.

3. PROBLEM FORMULATION

In this work, the simple instinct of feed-or-flee will be examined. The environment that the agents reside in consists of two consumable items: food and poison. The food sources have a positive effect on the agents, while the poisons have a negative effect. Additionally, multiple agents co-exist in the world, competing for the food sources. Finally, four outer walls will limit the environment's spatial boundaries. The agents will have to adapt a response to encounters with these walls, amongst other items in the environment.

The population of agents will live for a year in the environment with replenishing food and poison supplies. Each agent carries its own health level, whose value is directly related to its movements, and food/poison consumption. This health indicator directly translates to the agent's fitness value, which will be introduced in a later section.

4. APPROACH

The implementation of the problem is separated into three sub-components: 1) environment, 2) agent representation, and 3) genetic algorithm. The following sections give a detailed description for each of these parts.

4.1 Environment

The environment consists of a 50 unit by 50 unit grid, where each grid space is occupied by either an agent, a food source, a poison source, or an empty space. Only one item can occupy a grid space at one time. Figure 1 shows four scenarios in which the food (designated as a group of three green circles) and the poison (red circles with X's) exist in different amounts. The agents are denoted as black circles with a line segment indicating which direction it is facing.

The consumable items (food and poison) remain at a constant number at all times. Their placements, however, are not fixed. Food grows in randomly distributed clumps, while poison regenerates in a uniform random distribution across the grid. Each environment is characterized by its food and poison source counts. These values can vary in various combinations of quantities. A *plentiful food* scenario has a large portion of the world populated by food items. *Moderate food* scenarios have patches of food clusters and empty areas scattered throughout the landscape. Worlds with equal food and poison are often characterized by a uniform random distribution of poison amongst small groupings of food. *Plentiful poison* scenarios are riddled with poison sources with a very sparse amount of food scattered throughout.

A *stable* environment describes the situation where the food and poison quantities remain constant from generation to generation. A *dynamic*, or unstable, environment occurs when these consumable counts vary drastically between generations. The effect of environmental stability on behavior

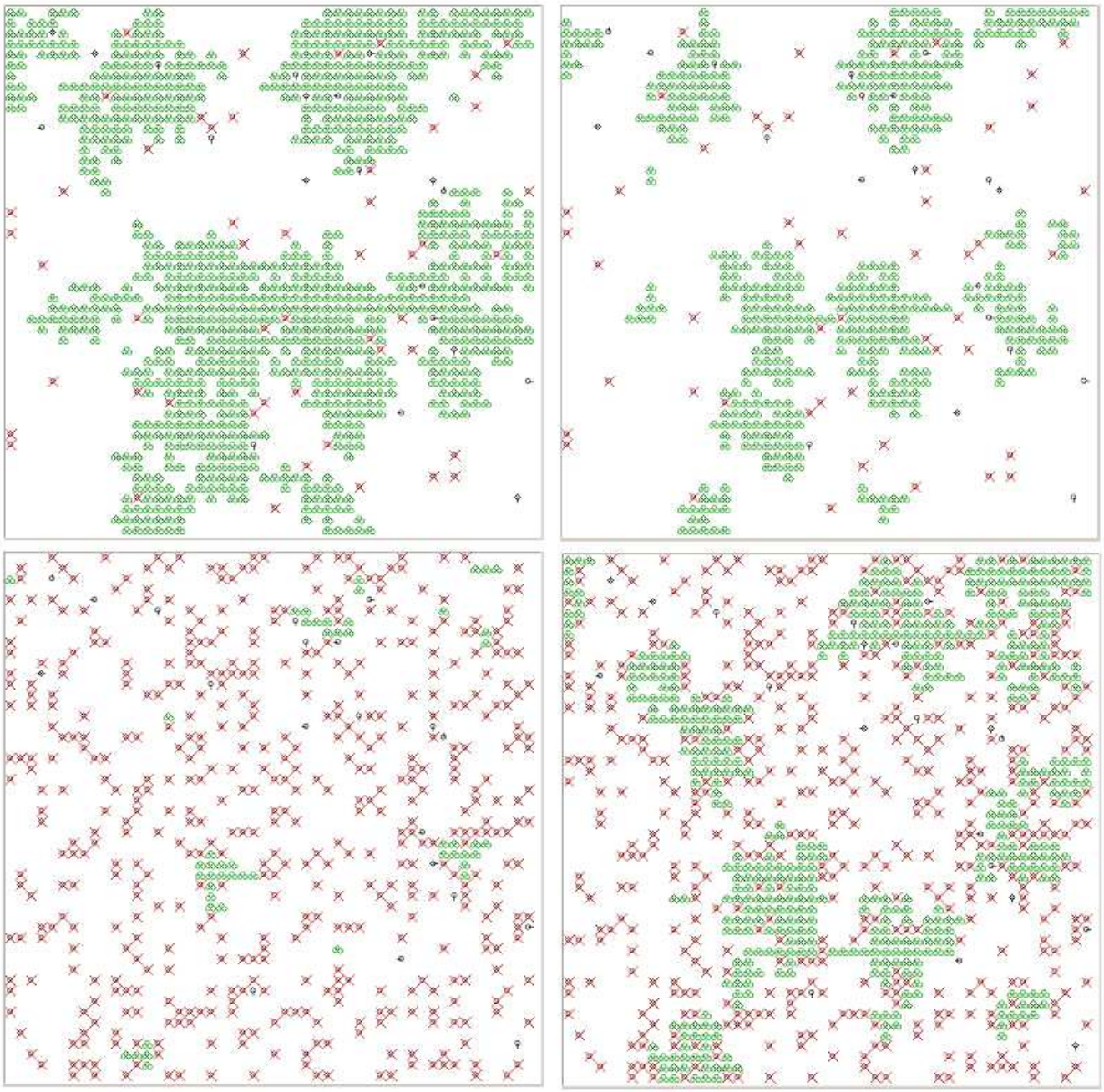


Figure 1: Enviroment scenarios (clockwise, starting from top left): plentiful food scenario, moderate food scenario, equal food and poison scenario, and plentiful poison scenario. Food is designated as a group of green circles, poison sources are the red X'ed out circles, and agents are the black circles with directional line segments.

evolution is examined in the experimentation section of this paper.

4.2 Agent Representation

Unlike the consumables, agents may roam freely throughout the environment. They cannot move further than the outer limiting walls of the world, nor can they move to any space occupied by another agent. An agent faces a cardinal direction (north, east, south, west), and its only sensing capability is the ability to see what item is immediately in front of it (empty space, food, poison, other agent, or wall). When an agent moves to a food or poison space, it *eats* that item. Eating food gives the agent 50 units of health, while eating poison reduces its healthy by 100 units. Agents all begin with a health of 100 units, and any movements (turn left, turn right, move forward, move backward) decrease their health levels by 10 units. When an agent encounters a wall or another agent occupying a space that it must move to, the agent simply remains at its current grid space without being assessed a health penalty.

Agent behavior is defined as a finite state machine (FSM) of N-states. This representation is derived from Eck's artificial life demonstration [3]. The FSM is encoded as a string of real values in certain positions. For each state, there are five transitions to other states, corresponding to each of the agent's five possible inputs. Agent input consists of which item (empty space, food, poison, other agent, or wall) is directly in front of the agent at a certain time. A next state and a next move (turn left, turn right, move forward, move backward) is encoded into the behavior string. The number of states in these behavioral models is completely arbitrary. Figure 2 gives an example of an FSM-based behavior representation for a 16-state agent behavioral model.

As mentioned before, the framework is flexible enough to maintain an agent model with any number of states. All agents in a single population, however, must share the same number of behavior states. The encoded behavior string is all that is needed to dictate agent responses. State zero is the start state, and the environmental input drives the agent's next move. The FSM is clock-driven, where each time step triggers a lookup on the agent's behavior string.

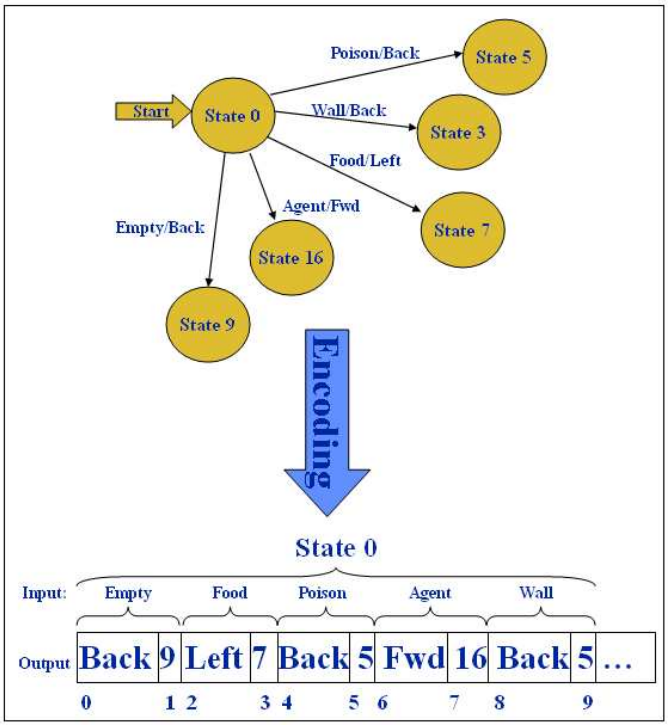


Figure 2: Behavior FSM encoding example. This figure shows a partial 16-state FSM and its equivalent form as a string of real values. There are two output values for each input (empty, food, poison, agent, and wall) for each state. These output values are: 1) the next move, followed by 2) the next state. Also note that the default state is the zero state.

4.3 Genetic Algorithm

Agent behavior is evolved using a traditional genetic algorithm [7]. A novel aspect of this genetic algorithm is that an FSM serves as the evolvable structure. Fogel [4] and Eck [3] both provide early examples of FSM evolution, while the latter provides direct support for the methods featured in this research.

Each agent population is subjected to a randomly generated environment for one generation, or 365 time steps. Regeneration of consumables occurs on a daily basis. After the generation concludes, fitness levels are assessed for the population and a reproduction system of crossover and mutation occurs to produce the next population. Fitness is calculated as the number of health units of an individual after the generation concludes, with an initialized health of 100. Two-point crossover happens at a 70% rate and mutation occurs at 0.1%. Parent selection is 4-agent tournament with

Table 1: Genetic Algorithm Parameters

Generation Length	365 Time Steps
Selection Method	Tournament, Size 4
Champ Selection Rate	0.9
Crossover Rate	0.7
Mutation Rate	0.001
Mutation Type	2-Point

a 90% champion selection. Table 1 summarizes the genetic algorithm parameters used.

5. EXPERIMENTS

Three experiments were defined to investigate two important aspects of this instinctive behavior system: 1) the optimal number of states for the behavior representation, and 2) the effect of the environment during behavioral evolution. Experiment 1 pertains to the former, while experiment 2 and experiment 3 concern the latter issue. The next sections describe each of these experiments.

5.1 Behavior Model Selection

The first experiment examines the optimal number of states for the agent behavior model. Five FSM-based behavior models were defined: 1-state, 2-state, 4-state, 8-state and 16-state. The 1-state behavior model was intended as a control; it was not likely to produce any reasonable agent response systems, due to its simplicity. The 16-state behavior model represents a very highly complex agent cognition, and it was featured as the default FSM size in Eck's artificial agent program [3].

Each behavior model was subjected to 2,000 environmental generations. 20 agents comprised the population size. A food count of 1,750 units and a poison count of 50 remained constant throughout the environment, constituting the plentiful food condition. Individual poison sources and food cluster placements were randomly distributed during each regeneration (both at daily and generational levels). The plentiful food scenario was chosen to allow all agents to have easily accessible food sources at any given time. This also prevented any behavior specialization, such as evolving specific strategies for limited resource situations. The average fitness and best fitness was recorded for each behavior model.

5.2 Effect of Static Environments

Experiment 2 shows how different environments affect the evolution of behaviors. Nine different environments, defined by different food/poison counts (10/0, 100/0, 1,000/0, 0/10, 0/100, 0/1,000, 10/10, 100/100, and 1,000/1,000) were tested for 2,000 generations using 20 agents in each population. The key ingredient in this experiment was the fact that all runs featured environments with a stable food/poison quantity from generation to generation.

This experiment compares and contrasts the eventual behavior responses, or instincts, that result from exposure to different environments. In essence, this effect can be characterized as the *shaping* of instincts by the environment. The overall champion behaviors of each of the nine environments were recorded and collectively analyzed.

5.3 Effect of a Dynamic Environment

The final experiment examines how a constantly changing environment affects behavior evolution. A population of 20 agents was placed in an environment for 5,000 generations. The addition of 3,000 extra generations allowed enough time to see any behavioral convergence trends.

A *dynamic* number of food and poison sources (random integral powers of 2 between 1 and 512) was assigned in each generation. A log of the average fitness and best fitness for each generation was maintained. A comparison of these statistics between runs with equal food counts (*i.e.* all generations with food count 512) was analyzed. The aim of this experiment was to emulate a constantly-changing environment and how such a situation affects the behavioral evolutionary process.

6. RESULTS

Data for each of the experiments were compiled and analyzed. The following section discusses these results and provides an analysis of each situation.

6.1 Behavior Model Selection

Figure 3 and Figure 4 shows the dominance of the 2-state behavior model over the rest of the field. It is observed that the 2-state version has a better overall average and best fitness and that it converges faster than the other behavior models.

The 1-state behavior model immediately flattens out, as expected. The 4-state behavior model performs second-best, following the trend that less complexity equates to better fitness values. The 8-state and 16-state models appear to overlap each other in both best fitness and average fitness. As a result of this experiment, the remaining experiments concentrate on implementing the 2-state behavior model. Figure 5 isolates the data recorded from just the 2-state behavior model generations.

6.2 Effect of Static Environments

Figure 6 exhibits the best-so-far graph for the second experiment. Figure 7 gives the resulting best behaviors from the different environmental configurations. It is observed that each scenario yields its own unique individual behavior. It is not sufficient to conclude, however, that different environments will produce different instincts. Analysis of the individuals reveals certain trends between agent inputs and responses. For example, the general trend for responding to a poison encounter is to either turn or back away - very rarely does the behavior direct the agent to go forward when presented with a poison source. A similar trend occurs with the food encounter, where the equal food and poison scenarios and the plentiful food environments evolve agents that will intend on eating the food.

The instinctive responses to the agent and wall inputs both usually results in a turning or a forward progression movement. The forward response would equate to a prevention of health penalty, since an agent cannot progress into a wall or onto another grid space occupied by another agent.

An interesting behavior trend is the tendency for the agent to move backward. This response is most prevalent in the

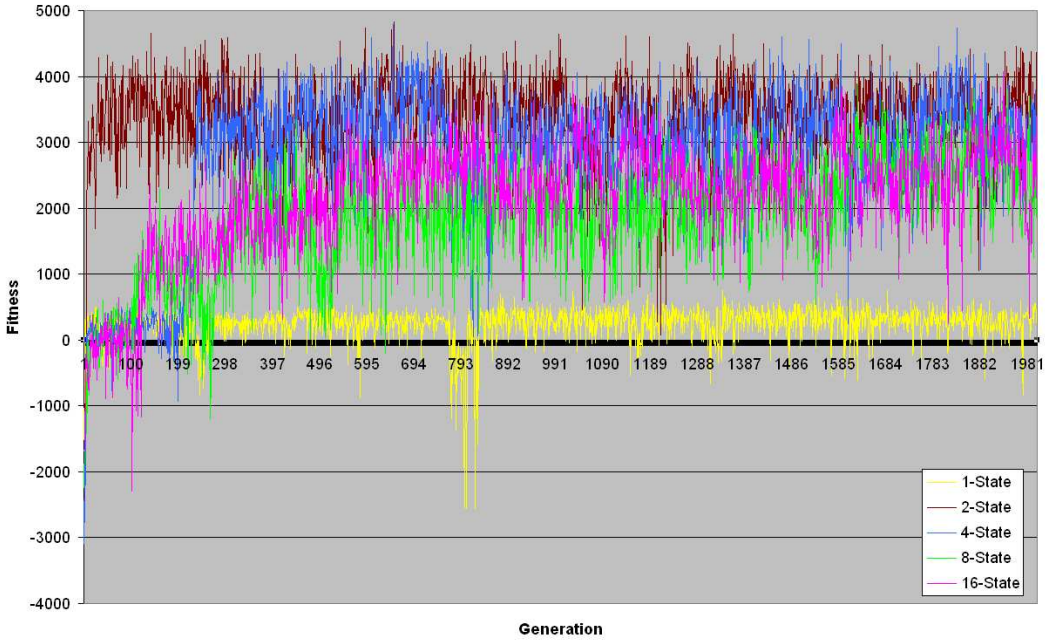


Figure 3: Average fitness for five types of behavior models. It can be seen that the 1-state behavior model performs relatively poorly, as expected. The 2-state behavior model produces the most effective agents, taking very little time to converge. The remaining behavior models all reach their convergence points by the 600th generation.

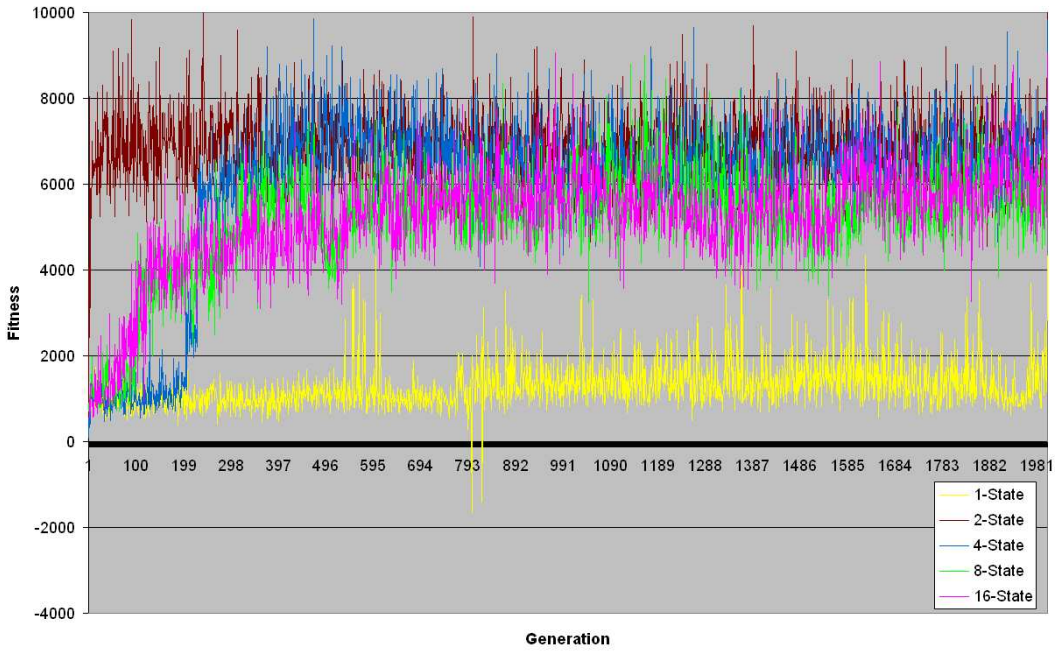


Figure 4: Best fitness for five types of behavior models. As seen in the average fitness graph, the 2-state behavior model proves most effective of the entire field. The 1-state model remains the control agent, and the remainder of the behavior models all peaked with best fitness values less than those of the 2-state model.

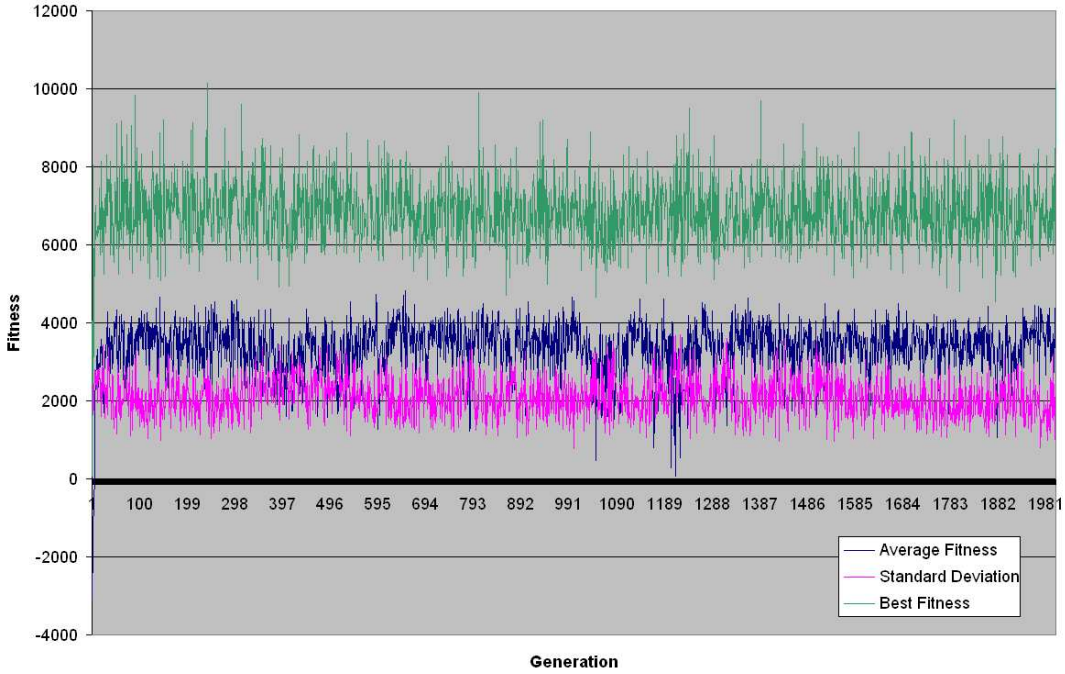


Figure 5: Evolutionary trend for 2-State Behavior Model after 2,000 generations. Almost immediately, all three curves (average fitness, standard deviation, and best fitness) converge to a linear asymptote. The true strength of the 2-state model, however, is its superior fitness values over models using larger numbers of states.

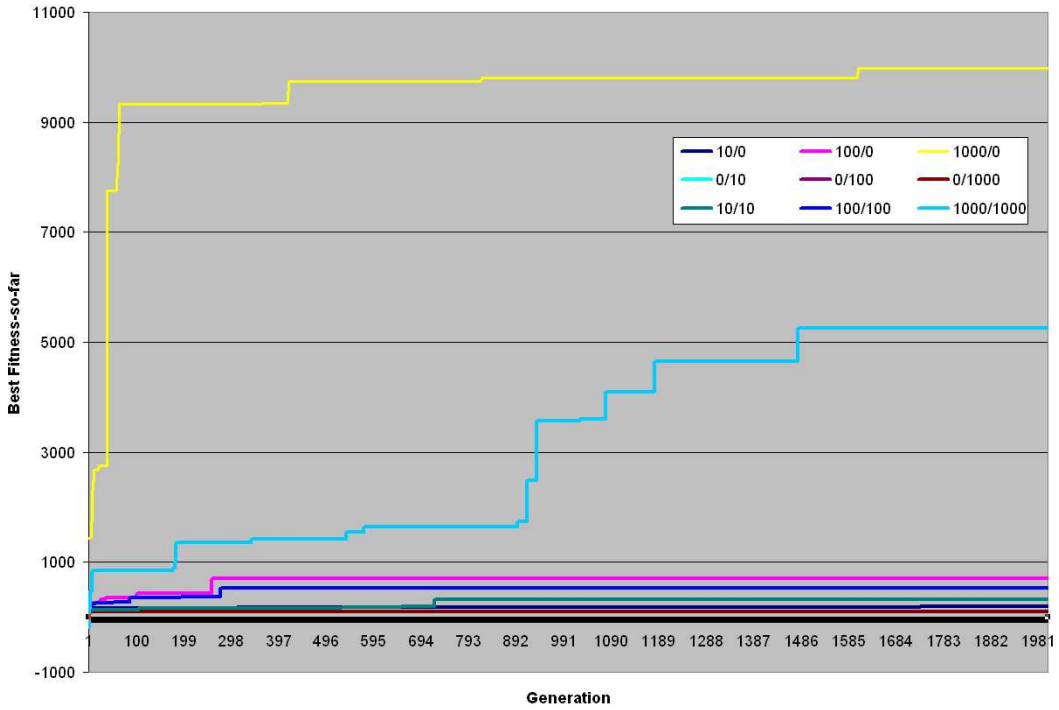


Figure 6: Best-so-far fitness values for nine different environment landscapes. Nine landscapes, defined by their food and poison counts (10/0, 100/0, 1,000/0, 0/10, 0/100, 0/1,000, 10/10, 100/100, and 1000/1000), exhibit different best fitness values as well as different convergence trends. The most plentiful food scenario, 1000/0, yielded the best overall fitness and reached this value very quickly. The 1000/1000, equal food and poison scenario required the longest amount of time to converge toward its best fitness.

Run		Current	Input									
			Empty Space		Food		Poison		Agent		Wall	
Food	Poison	State	Next State	Next Move	Next State	Next Move	Next State	Next Move	Next State	Next Move	Next State	Next Move
10	0	0	1	Backward	1	Turn Left	0	Turn Left	1	Forward	0	Forward
		1	0	Backward	0	Turn Left	0	Backward	1	Forward	1	Forward
100	0	0	0	Backward	0	Turn Right	0	Turn Right	1	Turn Right	0	Forward
		1	0	Forward	0	Backward	1	Forward	0	Backward	1	Turn Right
1000	0	0	0	Forward	1	Forward	1	Backward	1	Turn Left	1	Turn Left
		1	0	Turn Left	1	Forward	1	Turn Right	0	Turn Left	0	Forward
0	10	0	0	Turn Right	1	Backward	0	Backward	1	Turn Left	1	Forward
		1	1	Turn Right	0	Backward	1	Forward	0	Turn Right	0	Forward
0	100	0	0	Backward	0	Turn Right	0	Turn Left	1	Forward	0	Forward
		1	0	Backward	1	Forward	1	Forward	0	Turn Right	0	Backward
0	1000	0	1	Backward	1	Turn Left	0	Backward	0	Forward	0	Turn Left
		1	0	Backward	1	Turn Right	0	Turn Left	1	Turn Left	1	Turn Left
10	10	0	1	Turn Right	0	Turn Right	1	Turn Left	0	Forward	0	Forward
		1	1	Forward	1	Forward	1	Turn Left	1	Forward	0	Forward
100	100	0	0	Forward	0	Forward	0	Turn Left	1	Turn Left	0	Forward
		1	1	Backward	1	Backward	1	Turn Left	1	Forward	1	Forward
1000	1000	0	1	Turn Left	0	Forward	1	Turn Left	0	Backward	0	Turn Right
		1	0	Forward	1	Forward	1	Turn Right	1	Forward	1	Forward

Figure 7: Best 2-state behavior models for nine different environment landscapes. It can be seen that nine unique FSM's were generated after performing this experiment. Each of these behavior models, however, appears to follow certain general guidelines, such as avoiding moving onto poison and moving toward food.

empty space encounter, yet it is also an evolved response for food and poison inputs. Moving away from poison seems quite intuitive, yet moving away from food is not. From observing these particular agents in the environment, it appears as though the backward movement serves as a double right turn (or 180 degree turn) to search for items directly behind the agent. In fact, a lot of the evolved champion agents use the backward move as a means to explore its surroundings when food is present, or to use it as the primary means to consume food items.

6.3 Effect of a Dynamic Environment

In this experiment, 5,000 generations were run, where each generation was subjected to an environment with randomly assigned numbers of food items and poison sources. Figure 8 and figure 9 isolate the generations that have 1-unit and 512-unit food quantities, respectively. The generations are chronologically plotted and their average and best fitness values were examined. Similar data trends were retrieved for the remaining food quantity environments.

From each of these graphs, there is no evidence that behavioral performance improves during this dynamic environment scenario. The instability of the disruption-prone environment causes the evolutionary process to behave in a similarly unstable fashion. Essentially, the carried-over best fit agents from previous generations are rendered useless as parents for the following, newly generated environment. In essence, the dynamic nature of the environment causes each previous population to be as effective as a randomly generated population.

This experiment reflects some intuitive tenets of genetic algorithms. Since the environment landscape directly relates to the fitness evaluation of the population, constantly changing the environment would nullify the perceived fitness of previous populations. In nature, a similar effect would occur if the seasons changed on a daily basis. Animal species would not be able to evolve cold weather and hot weather behaviors in an efficient manner. In the case of this experiment, the food and poison levels varied so wildly that the agent populations could not effectively isolate which behaviors would be appropriate to pass onto its offspring as instincts.

7. FUTURE WORK

Improvements for this work would include further development of the graphical interface for the artificial life simulation. A real-time visualization of the agent populations in the environment was envisioned, but not fully realized.

A larger world and different population sizes would be beneficial to create more realistic environments. Another enhancement would be to add a third dimension to the environment landscape. Also, agents may be equipped with better sensors, allowing them to see a peripheral view of the landscape, rather than just see what is directly in front.

Additional instincts may also be added, where a predator-prey agent model could be implemented to develop insight into the fight-or-flight instinct (a behavior that is closely related to the feed-or-flee decision).

8. DISCUSSION

From the preceding experimental results, it is shown that a very simple behavior representation in the 2-state model can effectively emulate instinctive responses. The larger models, up to 16 states, appear to require a more complex evolutionary process to fully realize their optimal configurations. As predicted, the single state model proved to be insufficient for this application. The superiority of the 2-state FSM behavior model accentuates the idea that simpler solutions are often times the better solutions.

In terms of the environment's effect on behavior evolution,

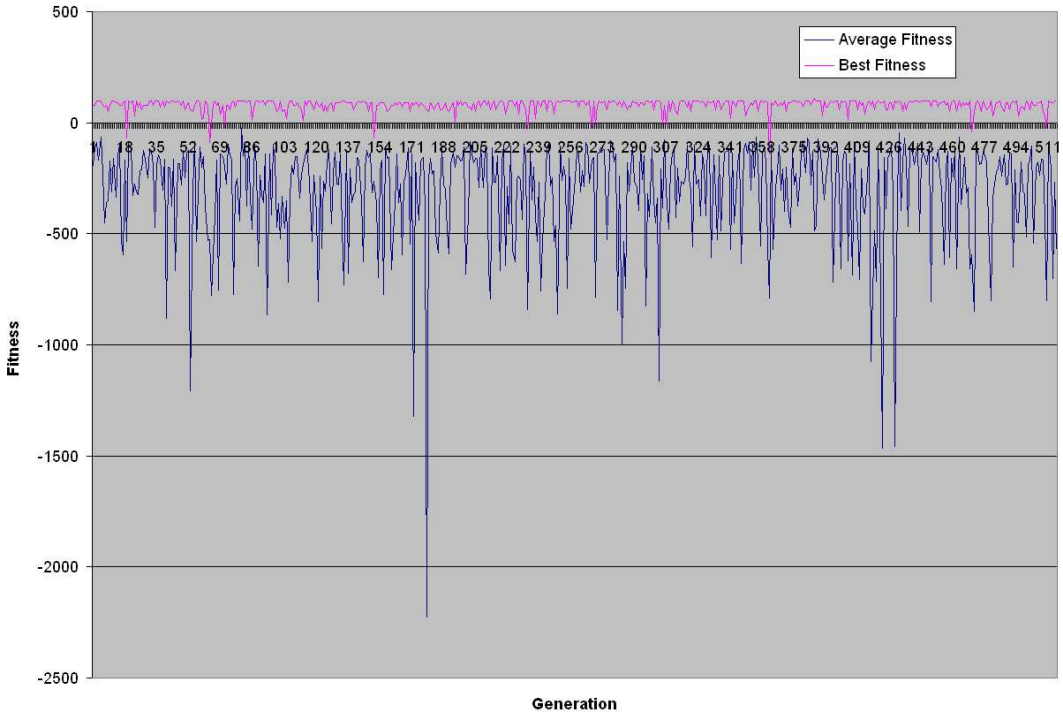


Figure 8: Best fitness and average fitness for all generations with 1 unit of food in a dynamic environment. No upward convergence patterns could be observed by this data set. The disruptive nature of the dynamic environment effectively nullifies any best fit individuals from the previous generation.

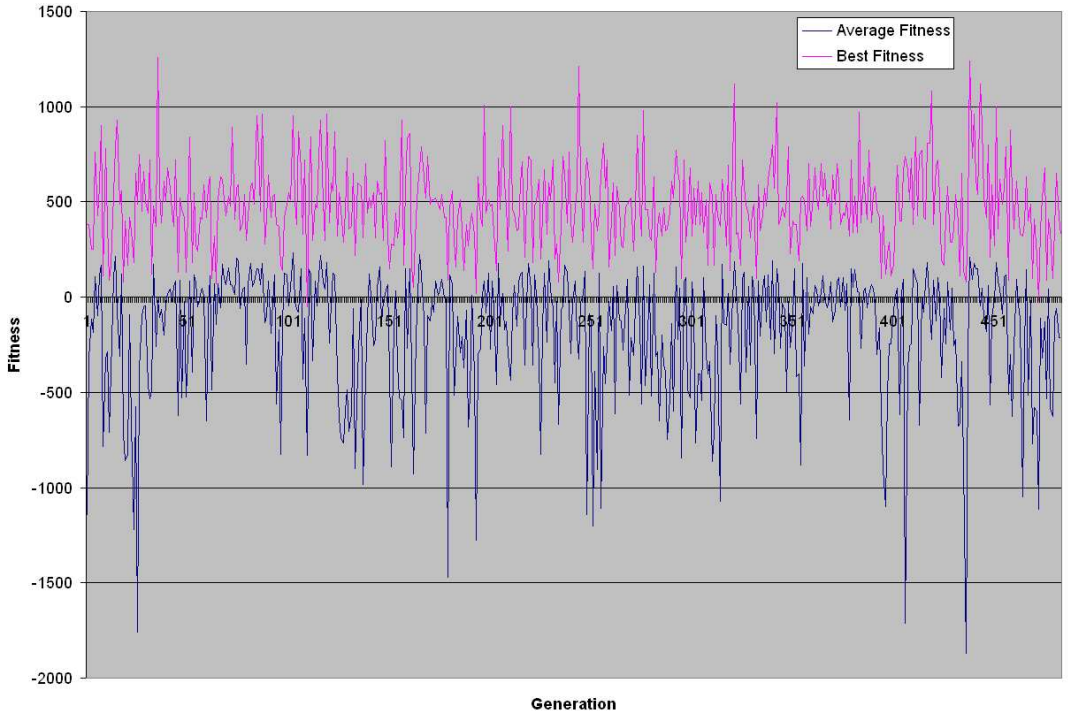


Figure 9: Best fitness and average fitness for all generations with 512 unit of food in a dynamic environment. As previously seen, these data points do not convey any remnants of evolutionary improvement due to the unstable nature of the environment.

it is asserted that instinctive behaviors can result from subjecting a species to a stable environment for a prolonged amount of time. Stability directly refers to the sustainment of the same environmental factors from generation to generation. An unstable, ever-changing environment will cause a species to evolve as if it is always being introduced to a new environment with each generation. Henceforth, it can be asserted that an instinctive response system is best evolved from an environment of predictable and stable factors.

9. CONCLUSION

Ethologists maintain that the evolution of behaviors in a species results in the existence instincts. This paper examines the emulation of behavioral evolution in nature as a means to produce instinctive behavior in an artificial agent. The two relevant issues involved in this endeavor are: 1) the proper behavioral representation structure, and 2) the effect of the external environment upon behavioral development. Experiments showed that a genetic algorithm that evolved the simplest behavior representation (a 2-state FSM) could create an effective set of instincts. Different environment landscapes results in different instinctive behaviors, although similarities between input responses were discovered. A dynamically changing environment was considered a detriment to the evolutionary development of instincts. Hence, according to this research, evolved behaviors may be perceived as instinctive if an agent population, with a simple behavior model, is exposed to a stable environment for a substantial number of generations.

Returning to the introductory thoughts of this paper, it was proposed that a generalized behavior set generation method could replace the hand-crafted agent modeling found in the today's modeling and simulation community. The work presented here suggests a brief step toward that realization, where the key ingredient to this autonomous agent building lies in the evolution of simple agents behaviors known as instincts.

10. ACKNOWLEDGMENTS

This paper was created for partial completion of the Evolutionary Computation course at the University of Central Florida. Many thanks goes out to the instructor of this class, Dr. Ivan Garibay.

11. REFERENCES

[1] R. S. Amant, S. P. McBride, and F. E. Ritter. Ai support for building cognitive models. *Proceedings of the Twenty-First National Conference on Artificial Intelligence (AAAI, Nectar track)*, pages 1663–1666, 2006.

[2] I. Brigandt. The instinct concept of the early konrad lorenz. *Journal of the History of Biology*, 38:571608, 2005.

[3] D. Eck. A demonstration of the genetic algorithm, http://math.hws.edu/xjava/ga/, 2001.

[4] L. J. Fogel, P. J. Angeline, and D. B. Fogel. An evolutionary programming approach to self-adaptation on finite state machines. *Evolutionary Programming*, pages 355–365, 1995.

[5] J. C. Giordano, P. F. Reynolds, and D. C. Brogan. Exploring the constraints of human behavior representation. *Proceedings of the 2004 Winter Simulation Conference*, pages 912–921, 2004.

[6] J. B. Haldane and H. Spurway. Imprinting and the evolution of instincts. *Nature*, 178(4524):85–86, 1956.

[7] J. H. Holland. *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. University of Michigan Press, 1975.

[8] K. Inoue and H. Kobayashi. Emotion model with evolution process. *Robot and Human Communication*, pages 338–342, 1997.

[9] R. Jones, J. Laird, A. Nuxoll, and R. Wray. Intelligent opponents for virtual reality trainers. *Proceedings of the Interservice / Industry Training, Simulation and Education Conference*, 2002.

[10] J. R. Kantor. A functional interpretation of human instincts. *Psychological Review*, 27:50–72, 1920.

[11] T. E. Portegys. Instinct evolution in a goal-seeking neural network. *The IASTED International Conference on Computational Intelligence*, 2006.

[12] P. Spronck, I. Sprinkhuizen-Kuyper, and E. Postma. Online adaptation of game opponent ai with dynamic scripting. *International Journal of Intelligent Games and Simulation*, 3(1):4553, 2004.

[13] K. O. Stanley, B. D. Bryant, and R. Miikkulainen. Evolving neural network agents in the nero video game. *Proceedings of the IEEE 2005 Symposium on Computational Intelligence and Games (CIG'05)*, 2005.

[14] N. Tinbergen. Social releasers and the experimental method required for their study. *Wilson Bulletin*, 60:6–52, 1948.

Learning invariant features in a set of similar fitness landscapes for speedup

Gautham Anil
ganil@cs.ucf.edu

Department of Computer Science,
University of Central Florida,
4000 Central Florida Blvd.,
Orlando, Florida-32816

April 12, 2007

Abstract

To increase search performance, domain information is commonly used to influence the bias of a representation and evolutionary operators for speedup. However, phenotypic information about the solution is never used, partly because in most cases information must be generated manually and thus is not practical.

We suggest and algorithm which, given a set of similar fitness landscapes (in the form of fitness functions), is able to extract, within the limitations of the representation, phenotypic information common to the landscapes which can be used to speed up search through a similar unseen landscape. It is also capable of extracting meta-information regarding searching through these landscapes.

This algorithm is then put to test on a set of simple landscapes satisfying the criterion. The information thus gathered is then used evolve solution of an unseen landscape and relative performance improvement is measured. The results are analyzed and conclusions are drawn regarding the effectiveness of the algorithm.

1 Introduction

An ordinary Genetic Algorithm[1] (GA) starts with a random initial population and evolves it using evolutionary operators such as mutation and crossover till a chromosome of required fitness is found. In a GA, it is possible for the scientist to optimize his representation and the evolutionary operators defined on it so that

the search is biased towards the solution (the field of genetic programming[2]). This means that the scientist tries hard to insert non-evolving *meta* information about the fitness landscape into the representation. Here, *meta* information is any information that affects the probability of producing chromosome *b* by repeatedly applying evolutionary operators on chromosome *a* (see [3] for a better explanation).

However, the experiment still starts with a random initial population. This means that there exists no *phenotypic* information regarding (the solution of) the fitness landscape in the initial population. There appears to be good reason behind it since to manually insert phenotypic information into the initial population is equivalent to making the search faster by solving a part of the problem in advance, manually. It clearly defeats the purpose of using computers to evolve the solution.

Sometimes, a scientist may be interested in only a small domain of problems. It might be possible that the fitness landscapes of these problems are *similar*. Similarity between two fitness landscapes depend upon the landscapes and the representation. It can be defined in two ways. In the simple case, if the solutions to the fitness landscapes have common portions in this representation, then the fitness landscapes can be said to be similar. However, in the hard case, they can be said to be similar (to a relatively lower degree) even if certain portions of the solution share biases (see Section 6).

Now, while it may be difficult to manually find phenotypic information, it might not be too hard to create a representative set of fitness landscapes from the domain of interest. Let us consider a situation in which we have available a set of similar fitness landscapes $F = \{F_1, \ldots, F_n\}$ over the same representation. We are interested in searching through an unknown similar fitness landscape U.

In this case, it might be possible to extract this common *invariant* information from the set of know fitness landscapes and use it in the initial population of the unknown landscape to achieve speedup.

In addition, consider the case where the evolutionary algorithm being used is able to force the chromosome to optimize meta information that it is capable of storing. Optimizing of the meta information improves the speed of search. If we are able to use this optimized meta information in the initial population, search performance might improve further.

We propose a simple algorithm called the *Island Algorithm* which given a set of similar landscapes F is capable of using any ordinary evolutionary algorithm as a sub-algorithm and extract if possible, phenotypic and meta information about F and use it to speed up search through an unknown landscape similar to those in F.

Further, we experiment with the island algorithm with GA as the sub-algorithm by applying it on a simple F on a simple representation with some evolvable

meta information. We then use the information gathered in the initial population and measure speed up achieved.

2 Island Algorithm

Given : E sub-algorithm

F set of n fitness landscapes

R Representation

1. Create n empty, populations (islands) and store them in I. Mark them inactive.
2. Mark some of the islands active and initialize them with random members of R.
3. Do till stopping condition (usually just loop count)
    1. For each active I_i in I, do E on members I_i with fitness function F_i until solution is found.
    2. $P = \sum_i^n I_i$ (add all members of the (active) islands to P).
    3. Remove the population of some islands and mark them inactive.
    4. Activate some inactive islands. To fill them with chromosomes, select parents from P (instead of random population).
4. Perform any information extraction procedure necessary on P (usually P will be used as starting population).

Here, at all times, inactive islands will be empty and no evolution will be performed on them. At all times at least some of the islands will be active.

3 Experimental Setup

Sub-Algorithm

To run the island algorithm a sub-algorithm is needed. We chose the standard GA as the sub-algorithm. The detailed configuration of the GA is given in Appendix A.

Representation

The chromosome was chosen as a sequence of real numbers which act as genes. In addition, each gene in the sequence was associated with a real number denoting the probability of this gene being mutated. These real numbers were chosen from the fixed set {.0001, .0005, .0025, .0125, .0625, .1, .3125} for

simplicity. The evolutionary operators on this chromosome were mutation and crossover.

Mutation started with first deciding whether to mutate mutation rates or the genes. If the mutation probabilities were to be mutated, then each gene was individually chosen for mutation based on a meta-mutation probability. If chosen its mutation probability was randomly either incremented or decremented to the nearest larger or smaller value respectively from the set of possible mutation probabilities.

If the gene values were to be mutated, it involved two steps. In the first step, a gene was chosen with probability proportional to the mutation probability. The value of this gene was mutated using a Gaussian function. In the second step, for each gene, based on its mutation probability, a decision was made whether to mutate it or not. Mutation was performed as before. This ensures that at least one gene mutates if the mutation function is called.

Crossover is the standard 1-point crossover [1]. The only extension is that the mutation probabilities are also crossed over along with the genes.

For this experiment, the length of the chromosome was 9.

Set of Fitness Functions

To keep things simple and run-times low, a simple hill climbing fitness landscape was needed. We used number-match fitness landscape which gave the hamming distance of the chromosome to a predefined target sequence.

The set of fitness functions consisted of number-match landscapes with different target sequences. However, all the target sequences shared the first six values. They differed only in the last three. The last three values were selected randomly at the beginning of the experiment from the range [-10,10].

Island Algorithm Settings

The island algorithm used 8 islands. At any time, exactly 3 was active. For easy book keeping, instead of changing islands every time they solved, it was done every 100 generations of the sub-algorithm. During every island changing step, exactly 2 islands were chosen for deactivation from the active 3 islands. Then, exactly 2 islands were chosen for reactivation. Each Island had a population of 30.

4 Experiments and Results

Case 1: As described

When the algorithm was run as described in section 3, it was noted that island

algorithm easily detected the similarity among all the first 6 values of all the evaluators. The population mostly differed only on the last 3 values. However, the meta information was not as good as hoped. Over successive runs, we found that there was only a small tendency for the first six genes to have low mutation probabilities and the last three to have relatively higher mutation probabilities. The distribution improved if the meta-mutation probability was increased. However, increasing of the meta-mutation probability reduces the influence of the mutating probabilities of the final population on the performance over the unknown fitness landscape. If the values were reduced, and the loop-count at step 3 set to large values, still the results produced were inconclusive.

Thus, we concluded that the selection pressure applied by the sub-algorithm (GA in this case) towards optimizing the meta-information in the chromosome is not strong enough to be able to use the meta-information directly from the final population.

Case 2: Modified Chromosome and meta-information extraction

To make the effect of selection pressure of GA more prominent, instead of using a mutation parameter for each gene in the chromosome, only two were used. One for the first six and the other for the last three. The idea is the since now there are lesser values to optimize, they might converge better. In addition, instead of using only these values from the final population, a counter was kept for each possible mutation probability value for both the probabilities. This is sufficient to give the complete distribution of values chosen during the entire island algorithm.

We also kept track of the number of generations an island spent looking for solution. This was done by cumulatively counting the number of islands which found solution at every generation of the GA. From this statistic, we find that on average, an island spends 38 generations looking for solution. Note that there are 3 islands and thus a population of 3 * 30 = 90.

The statistics gathered about the mutation value frequencies are in Table 1.

Mutation Value	**0.0001**	**0.0005**	**0.0025**	**0.0125**	**0.0625**	**0.1000**	**0.3125**
First-six mut. prob.	6807180	7670765	6054676	3891171	2565015	801005	110095
Last-three mut. prob.	3946	225120	421069	1912844	7466354	6853528	11017046

Table 1: Mutation probability frequencies

Already we can see that first-six spent more time on lower values while last-three spend more time on higher values. When we take a weighted sum of the mutation rates, we get,

First-six = 0.0123472541

Last-three = 0.170524441

Testing: Standard GA with random initial population

Next, we change the island population size to 90 and total island count to 1 simulating a single GA run. We remove the meta-information from the chromosome. Gene mutation probability is set to a fixed .1. We start with initial random population and do 10 runs. We find that the average number of generations needed to find a solution is **126.7**.

Testing: Standard GA with phenotypic information

Here, instead of using initial random population, we use the champion from case 2 having removed the meta-information. Doing 10 runs, we find that the average number of generations needed is **45.8**. This amounts to an improvement of **176%**.

Testing: Standard GA with phenotypic and meta-information

Here, we add the meta-information back to the chromosomes. Also, we initialize the meta information with the values calculated in case 2. Also, the meta-mutation probability is set to 0 to take full advantage of this mutation probabilities. We find after 10 runs that the new average number of generations needed is **34.3**. This is **33%** increase over phenotypic information only and **270%** improvement over standard GA.

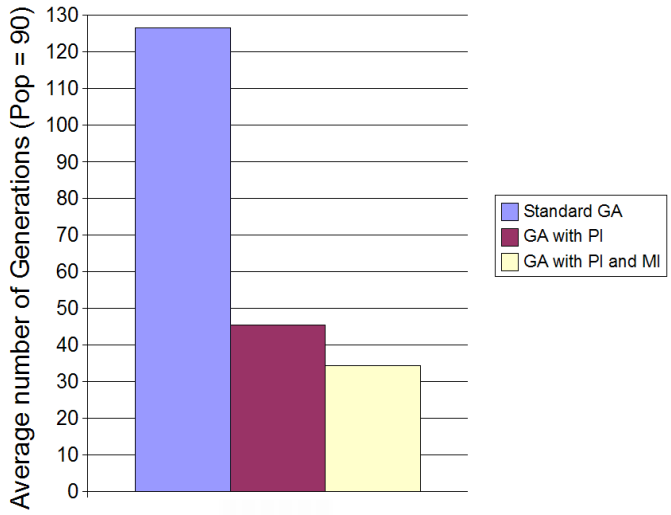


Figure 1: Performance in Test Experiments

5 Conclusion

We explain how a scientist while optimizing meta information about the search space embedded in the representation's bias, is unable to find and use phenotypic information about the solution.

It is suggested that given a set of similar fitness landscapes, we might be able to find common phenotypic information automatically which could be used to speed up search for the solution of a similar landscape.

An algorithm is then suggested which uses an existing evolutionary algorithm to find this common phenotypic information and any meta-information about the landscape that can be extracted. This algorithm is then applied to a simple case that satisfies the requirements. The information gathered is then used in a GA and performances are compared.

It is found that in this application of Island algorithm with GA as the sub-algorithm, we get notable performance improvement over standard GA. However note that the exact values of improvement are of no consequence as they can easily manipulated to a large extent simply by changing the parameters of the experiment.

What is important is that given a set of similar fitness landscapes, the algorithm managed to extract some phenotypic and meta information. The increase in other domains too would largely depend upon the representation used, the similarity of the fitness functions, the quality and quantity of selection pressure applied by the sub-algorithm etc.

6 Disadvantages and Future Work

Find an algorithm which applies more selection pressure than the standard GA on optimizing meta information and integrate this new algorithm into the island algorithm. Since the island algorithm maintains the population between successes, enhance the meta information by using this repetitive-success. This is like using hindsight to make better decisions. Hindsight information is only available if success is achieved at least once which makes it impossible to use it in an ordinary evolutionary algorithm.

The island algorithm also needs to be tested on a more difficult set of fitness landscapes with less intuitive invariant information.

The island algorithm as such is able to detect common phenotypic information only if they are the same. If they are only close, say in the range of 4 to 6, the algorithm will not decide on 5 which is the optimal phenotypic information. It will instead present the last seen value. This means that the island algorithm requires F to satisfy the simple similarity requirement. It is to avoid this "last-seen" effect that the island algorithm has multiple islands active at a time. But as such they alone are insufficient to encourage such optimal behavior. Thus, modifications must be researched to enable it to extract optimal phenotypic information even with different but similarly biased values. In other words, enable it to extract phenotypic information even if F satisfies only the hard similarity requirement.

At the moment, the island algorithm can only work with representation with

fixed length. This means that the dimensions are named. The algorithm needs to be improved to handle unnamed dimensions.

7 Appendix A – GA (Sub-Algorithm) Configuration

Population size	30 (Island size)
Selection criterion	Tournament (k = 2)
Probability of mutating	1 (Every member has been mutated)
Probability of crossover	1 (Every member is product of crossover)
Mutation type	Gaussian (σ = .1)
Elitism	None

8 References

[1] Holland, John H , *Adaptation in Natural and Artificial Systems*, University of Michigan Press, Ann Arbor, 1975

[2] J. R. Koza. *Genetic programming: A paradigm for genetically breeding computer population of computer programs to solve problems*. MIT Press. 1992.

[3] Toussaint, M. Demonstrating the evolution of complex genetic representations: An evolution of artificial plants. In: *2003 Genetic and Evolutionary Computation Conference (GECCO 2003)*. (2003)

Optimal Design and Rehabilitation of Water Distribution Networks using Evolutionary Computation Algorithms: a Literature Review

Abhishek Das
Dept. of Civil and Envr. Engg.
University of Central Florida.
214 Engineering Building 2, University of Central Florida
Orlando, FL 32816-2450
Phone: 001-407-823-1056
abhishekdas.das@gmail.com

ABSTRACT
In this paper we discuss the work done in the field of optimal design and rehabilitation of water distribution networks (WDNs) using evolutionary computation (EC). WDNs form a vital part of infrastructure, constructed to satisfy water demand at consumption points (nodes). The system must also satisfy pressure requirements at the given nodes. With the passage of time the existing WDNs will degrade in performance and need to be rehabilitated. With growth in demand new WDNs become necessary and old ones need to be extended. Conventionally the design of new WDNs or the up gradation of the existing ones had been done on the basis of engineering judgment. Though the requirement of engineering judgment is still essential, new computation tools augment the work of the engineers to a great extent. Optimizing the WDNs is not a small task, as the complexities are many and effort is always on to find new algorithms to optimize the design or rehabilitation work. Here we will be specifically discussing the use of evolutionary algorithms (EA) that have been used in the field to augment the engineer. The discussion presented here is a qualitative comparison.

Note: The references list contains only those that are cited in the text, numbered in the order in which they are first cited.

Keywords
Water distribution networks, evolutionary algorithms, cost minimization, reliability maximization.

1. INTRODUCTION
WDNs form a vital part of infrastructure, constructed to satisfy water demand at consumption points called nodes. The system must also satisfy pressure requirements at the given nodes. With the passage of time the existing WDNs will degrade in performance and need to be rehabilitated. With growth in demand new WDNs become necessary and old ones need to be extended. Conventionally the design of new WDNs or the up gradation of the existing ones had been done on the basis of engineering judgment. Though the requirement of engineering judgment is still essential, new computation tools augment the work of the engineers to a great extent. Optimizing the WDNs is not a small task, as the complexities are many and effort is always on to find new algorithms to optimize the design or rehabilitation work. Earlier traditional optimization methods were used for the purpose, but problems like handling of discrete decision variables, getting trapped in local optima led researchers to the use of EAs. The EAs can handle not only discrete variable optimization but are also capable of not falling at deceptive points. They are robust and can explore the entire search space in lesser time than any traditional optimization or search process. The late 1990's saw the use of EAs in the field of water resources engineering and since then various EAs have been tried and tested successfully.

In section 2 we will be discussing about the uncertainties and what objective functions are sought to be optimized in any water distribution network design problem. Section 3 will be dedicated to the discussion of why EAs are better at handling the problem than the earlier methods. Section 4 will include the discussion the various EAs used. Section 5 will be the conclusion.

2. UNCERTAINTIES AND THE OBJECTIVE FUNCTIONS
A typical WDN consists of pipes laid out, with constraints of existing roadways and buildings. Finding the shortest path to a certain node from the source of water is not always possible and is actually never the goal. The aim is more or less always to find the least cost design [1] layout keeping in view the demand at all the points. The pressure demands through out the system have to be maintained also. And also not forget we have the crucial uncertainties [2, 3]. The uncertainties are generally of the following types: 1) hydraulic uncertainty, 2) mechanical uncertainty, 3) demand uncertainty and 4) hydrologic uncertainty. Hydraulic uncertainty pertains to demands not being met at certain nodes because of the long distance from the source or being at a higher elevation than the source. Though we cannot do much about the source of the water but we can definitely take action in the form of putting in pumps at the right places. So that the when the pressure drops significantly at places due its location or distance, we can ensure reliability. Due to hydraulic uncertainty though the demand is satisfied at all times, some of the nodes are not served. Mechanical uncertainty could arise due to any sort of mechanical failure in the system like the breakage of pipes, bursting of valves or breakdown of pumps. Mechanical uncertainty leads to the condition in which demand is satisfied at all nodes but not at all the times. Demand uncertainty rises due to various causes like sudden rise in the population of a particular

region which is in turn related to increase in business in that particular region. This could also be if the forecast has not been very properly modeled. Hydrologic uncertainties are owing to problems at the source of the water, e.g. less than average rainfall can lead to a reduced amount of water available in the dry period. Though supply can be available at all times to all nodes but the demand will not be satisfied. So keeping in mind that we have these kinds of uncertainties plaguing us all the time for which we have limited control, the design or the rehabilitation work should be able to take into account the above mentioned uncertainties and make the system reliable. Resilience [4] is the ability of the system to come back to its normal state after a momentary or brief departure. So network resilience [5] could be a good measure of the reliability of the system, which is one of our objectives. But since WDNs are expensive infrastructure, the finding of a least cost design is but an inevitable objective for most of the cases under study. While optimizing these objective functions the usual equations of the conservation of flow and energy has to be satisfied. The flow equation to be satisfied is that the difference in the amount of water flowing into a node from nearby links and the amount of water going away from the node is equal to the demand at the particular node. The pressure constraint that has to be satisfied is that there must be a minimum pressure head at the node. A pressure violation is said to occur when the pressure drops below the minimum.

3. WHY EVOLUTIONARY ALGORITHMS?

Traditional optimization methods have earlier been used for the design of WDNs before the use of evolutionary algorithms became popular. And with the advent of faster computers intensive computations was no longer an issue to be considered. But the old methods of optimization had some serious drawbacks. The system under consideration is non-linear and the old techniques rely heavily on linear simplification, which though makes the calculation simpler leads to unacceptable results. It also assumes that the system is deterministic but where as it is not. In the problem at hand diameter of pipes that are to be laid out are the decision variables in most cases. Since they are manufactured in large numbers and themselves are expensive units, they are only produced in certain sizes. In other words the decision variables are discrete variables. Old methods could not handle discrete variable. They treated the decision variables as continuous and the results would always have to be rounded off to the closest available discrete pipe size. This can lead to not only sub-optimal solutions but also to infeasible solutions. An infeasible solution is one where the constraints are violated. When multiobjective optimization is attempted multiple solutions cannot be found in one run [5]. So we need to have multiple runs for obtaining multiple solutions and also there is no guarantee that varied solutions will be obtained. Some traditional methods also cannot model the use of pumps, valves and storage tanks which are very important components of the WDN [6].

EAs like genetic algorithm (GA) or memetic algorithm have the advantage of being able to handle discrete variables. It does not matter whether the system is linear or non-linear. The ability to represent solutions which can be mapped to real-life situations, play around with the solutions, evaluate them as we go based on certain fitness function is the key to the robustness of the GA [7] or more generally EA. And single run with enough number of generations so as to allow for convergence to take place is sufficient to give solutions for multiobjective functions. EAs which are stochastic tend to provide better solutions for water distribution systems which throw light on the nature of the problem itself. They can very efficiently handle pumps, valves and storage tanks. They have been proved to have performed better than deterministic optimization techniques [8] in the optimal design of WDNs.

But it is also to be noted that EAs are also not great at handling constraints. So in all the cases the constraints are solved by an external hydraulic network solver [9]. The network solver takes solutions evaluated by the EA and then sees if the solution is feasible or not. It gives the feedback then back to the EA. This feedback is crucial because in most cases a penalty function is evaluated based on whether the constraints have been evaluated or not. This penalty value is then included in the computation of the fitness value of the solution.

4. EVOLUTIONARY ALGORITHMS USED IN THE DESIGN OR REHABILITAION OF WATER DISTRIBUTION NETWORKS

The various EAs that have been used in the given field problem will now be discussed. The goal of the project here has been to understand the various EAs implemented and how effective they have been in comparison to other algorithms. One of the short comings that we have observed here is that some researchers come up with certain modifications to the simple GA and try to beat it down. But in certain cases the authors have tried to improve upon modified GA. This is kind of interesting to study as it sort of exposes the delicate aspect of the algorithm. Through out the study we realized how difficult it is to get a perfect optimization technique for a real world problem. In these high dimensional problems there is always scope for improving the optimization technique and this is precisely what has happened in the given cases under study. Researchers have used the simplest of the evolutionary algorithms to the relatively more difficult concepts of evolution to optimize the networks.

The various EAs to be discussed in the following subsections are the following: 1) Single objective GA, 2) multiobjective GA, 3) hybrid GA, 4) messy GA, 5) Ant System Algorithms, 6) Shuffled Complex Algorithms.

The New York City water system and the Hanoi Network are the two classical networks in literature. They have been tried and tested on each and every type of algorithm discussed in the following sections. The use is primarily because they have been traditionally benchmark problems. So it makes more sense to compare the results of the new algorithms with the older ones. In addition to that the complexity of the networks is high but not too high. So dealing with these networks gives enough challenge and computational ease at the same time.

4.1 Single objective genetic algorithm

Most of the initial work of the application of EA in this field has been the implementation of the single objective GA. The objective most of the time has been to minimize the cost, except for few like [10], where reliability is the sole objective to be optimized. When the goal is single objective it is argued that resulting structure of the WDN will be more tree-like. This could

be explained in very simple terms as follows. If we consider reliability then that means that we have to have loops in the system which will be redundant and will only come into picture during the failure conditions and help in the system resilience. But if only cost minimization is the goal then those extra loops no longer come into the picture and the developed structure is more tree-like [1]. The representation has been binary [11] as well as real. The problems with the binary representation have been discussed by [12] and the researchers have advocated the use of the real coding to be used for representation. The binary representation is very easy to understand but has the problem of the Hamming cliff effect [13]. So some authors have used Gray coding [1] instead of the binary code to overcome the problems. Real coding sorts out the problem of redundancy associated with binary coding. It also allows for more variety of combination and mutation operations to be implemented [14]. As mentioned earlier a penalty function is included to account for pressure infeasible solutions. The fitness function is directly proportional to the reliability and is inversely proportional to the cost of the system. For reliability maximization problems it has been observed that it is better to look at the network reliability rather than critical node reliability [10]. In the latter case there is an inherent problem of over estimating the network reliability.

4.2 Multiobjective genetic algorithm

In a typical implementation of the multiobjective GA in the design of WDN, we observe that cost minimization and reliability maximization are the two goals. These two goals intuitively tend to play against each other. But it has been shown that simultaneous optimization of the cost and reliability resulted in high quality solutions (Parerto Fronts) and also inclusion of a third objective e.g. minimum surplus pressure head also improves the solutions [15]. [4] have used non-dominated sorting algorithm (NSGA) for optimizing the two objectives. In NSGA a ranking method is used to emphasize current non-dominated and a sharing parameter is used to maintain diversity in the population. Crossover and mutation are carried out. Selection of parents from different non-dominated fronts is done by tournament selection. SPEA which includes elitism has been shown to perform much better than NSGA [16]. The developers of NSGA came up with NSGA-II which was an improvement on the original [17]. They decreased the complexity, introduced elitism and did away with the sharing parameter. This elitist MOEA was tried by [18] on two classical literature networks. [19] tried NSGA-II and Strength Pareto evolutionary algorithm 2 (SPEA2) on two classical networks. They found that though both generated Pareto sets but had incomplete solutions, thus were suboptimal. SPEA2 performed slightly better. The result is interesting as it shows that lot of work has to be done to locate optimal Pareto sets for high dimensionality objective spaces.

[20] & [21] have used multiobjective approach for cost minimization and energy savings in water supply by improving the operations of the pumps. Earlier it was not possible to optimize the performance of pumps using traditional optimization methods. But as we had mentioned earlier the optimization of pumps has been achieved successfully by multiobjective GA.

4.3 Hybrid genetic algorithm

The concept of hybrid GA is to use the “best of both worlds”. Local search methods are good at converging to optimal solutions quickly using the gradient information, but are not able to jump around. Thus they get trapped in the local optima [22]. GA on the other hand becomes less efficient when they find the near optimal solutions. A hybrid GA thus combines a local search with a GA. Based on the particular optimization problem under study it is decided as to how to combine the both. They are generally classified as two methods: the sequential global – local method and the embedded global – local method [23]. In the first stage of the sequential global – local method the GA is implemented to give us the near global optimal solution. Then the local optimizer takes the initial estimate and searches for the global optimum in the region of the near global solution. The embedded global – local method is an iterative process between the search operators of the global optimizer and the local optimizer. In some problems like in the case of two way pipe flow [23] there are complicating variables. The global optimizer is used to take care of the complicating variable and then the local optimizer is used to resolve the rest. Convergence to global solution is reached through iterations. In both the above methods the solutions are achieved much faster than the GA used alone and also results in some studies find that they give better solutions than the simple GA.

Now we will be discussing about another type of hybrid GA whose foundations is based on a different aspect of GA. GA begins with a randomly generate solution set and then after lot of generations of fitness evaluations comes up with the best solutions. Though a bad start doesn’t imply bad solution, but it may affect the optimality. Bad start sometimes hinders the search for the global best [24]. This could be because we do not have any prior knowledge of the environment and if we could get that information before starting the process then it is believed that better solutions could be achieved. Cellular Automata (CA) has been widely applied to spatially distributed problems like simulations of traffic flows. A CA consists of interconnected set of nodes that uses certain rules to update the state of the nodes. The rules are problem dependent and updates are done in parallel. The CA is only concerned with implementation of rules at the local level. CA has features like parallelism, localist representation and homogeneity [25] and can reach close to Pareto optimal solutions in small number of simulations. Then the results of the CA can be used to seed some GA e.g. NSGA-II, which can reach global optimum [26]. The method is kind of reverse of the sequential global – local method. Here the local optimizer works first and then the global is used.

The hybrid GA has been used for both single and multi objective functions optimizations.

4.4 Messy genetic algorithm

It is based on the concept that the best solution will be found in the solution region containing a high proportion of good solutions. The simple GA has fixed length strings. The messy GA has variable length strings that grow over generations. Not only have they grown over generations, but they also have different lengths in a given populations giving it a messy look and hence the name. This versatile string length is the key to the messy GA [9]. The variable length string along with the cut and splice operators helps

in identifying good clusters of string bit patterns that are contained in good solutions. The genes are identified as a pair of gene locus and gene value. Gene locus is the sequential order of a gene bit in the full length string. The messy GA imitates the evolutionary process in two nested loops. Each cycle in the outer loop calls upon an initialization phase and an inner loop that has a building block filtering phase and a juxtaposition phase. The larger the population size the higher is the chance that all possible good building blocks will be present. The building block filtering process identifies better fit shorter strings and randomly deletes genes and is done till a desirable length is reached. In the juxtaposition phase the resulting strings of the building block filtering phase are combined to form new strings. The cut and splice operators are called during this step and helps in splitting or combining strings. The cut probability is higher for string of longer length. In the early stages of the juxtaposition phase the probability of the splice operator to be called is higher as the strings are of shorter lengths. Cut and splice operator when called upon to act on two strings of same length in succession may act like a crossover operator. So this shows that cut and splice operator have more versatility than the standard operators. Due to these features of the messy GA they have been found to perform better than the fixed length GA in standard networks.

The structured messy GA used by [27] and [28] partially uses the capability of the messy GA. In these the string length grew only over the generations so there was no messy population. This allowed for the use of the standard crossover and mutation but could not use cut and splice operators. So the features which made messy GA more powerful were not used. So they could not be as efficient as the messy GA.

The messy GA and the structured messy GA have been used typically for cost minimization purpose.

4.5 Ant system algorithms

Ant colony optimization (ACO) is a probabilistic optimization technique developed for solving computational problems which can be reduced to finding good paths through graphs. The ant algorithm for discrete optimization was developed by [29]. They are inspired by the foraging activity of the ants in finding paths from the home to food and back. In the real world ants travel randomly and after they find food they return home by laying down pheromone trails. Other ants who find the trail will no longer travel randomly but instead follow the trail to find food. The in turn will also lay pheromone trails thus reinforcing the path. The pheromone evaporates with time, so longer paths will have reduced attractiveness in comparison of shorter paths. This is nature's own way of preventing stagnation or in other words preventing traps at local optima. In ACO the behavior of real ants is mimicked with simulated ants. The ACO is nested loop algorithm with the outer iteration equivalent to the generation of GA. The inner cycle is equivalent of the evaluation of solutions in GA. Number of cycles is also the number of ants generated. The ACO process is governed by the ground pheromone intensity. Since each solution is a trial solution the pheromone is updated after each cycle. The iterations or the outer loop continue till a stopping criterion is reached. The stopping criterion for the given problem at hand is that the demand at all nodes must be satisfied at all the times [30]. The ACO was found to be performing better than the standard GAs in terms of faster convergence to global optimal solution. In the ACO the pheromones are laid on the ground and the ants are just to make decisions as to where to go next after they reach a certain point. So we see that changes are actually made to the environment and thus they become powerful optimization tools in dynamically changing problems like the WDNs. Rank based updating scheme implemented ACO and Max – Min Ant System (MMAS) also appear to perform better than the basic ACO [31]. Iteration best Ant System also performs better [32]. The authors have also suggested modifications to parameters so that performance improves. MMAS have a dynamically developing bound on the pheromone trail intensity such that they remain within certain limits of the path with the greatest pheromone intensity. This gives each path a non-trivial probability of being selected and thus allowing greater search scope [33].

Primarily the algorithms have been tested on single objective functions, in this case cost minimization.

4.6 Shuffled complex evolutionary algorithm

The shuffled complex algorithm starts of with the combination of probabilistic and deterministic approaches, evolution of complex solutions, competition and complex shuffling. A population is randomly generated and is sorted in ascending order. The population is then partitioned in to complexes. Each complex is allowed to develop independently. Points within a complex produce new offspring which then replace the worst fit. The points in the evolved complexes are pooled together and then shuffled again. This shuffling causes the information to be shared among every member [34]. This is memetic evolution. The shuffling goes on till a terminating criterion is achieved. The criteria could be a pre-specified number of function evaluations or could be that the relative change in the function evaluation in the last m shuffling loops the value is within a tolerance level. The implementation of the above algorithm in WDN performed better than simple GA and some other Shuffling Algorithms. A serious disadvantage is that this algorithm cannot handle discrete variables.

Shuffled Frog Leaping Algorithm (SFLA) is very similar with a few changes [35]. In the former there was generation of new points which replaced some worst fit points. In the latter PSO is used for the local search inside a complex and the sorting is dome in descending order. Inside the complex in the SFLA the position of the worst frog is tried to be improved, where is in the simple shuffling algorithm reproduction takes place and the worst fit is removed. The SFLA can handle discrete variables. The algorithm is named so after the fact that frogs leap from one stone to other in a pond to get to the food and they communicate with each other to get the information about food. When the local performance cannot be further improved the complex shuffle. The process of shuffling is analogous to the set of cultures that were working isolated and now meeting and exchanging ideas. Stopping criteria are same as the simple shuffled algorithm. SFLA was found to perform better than traditional optimization techniques.

Shuffled complex algorithms have been used as single objective function optimizer in the WDNs design.

5. CONCLUSION

So far as we have studied the broad categories of EAs that have been used in the design of WDNs and realized that all of them have improved the design and search for the optimal solution. A quantitative comparison is beyond the scope of this work as it is a literature review. Since WDNs are costly infrastructures minimizing the cost is a by default objective. Apart from that maximizing the network reliability has been a major goal. Operation work like pump scheduling sometimes come up as an objective, but very rarely. Water quality as an objective also appears but again rarely. Single objective GA and multi-objective GA are the ones most commonly used as hey have been tried ad tested for longer period of time than others. The use of messy GA for serious optimization purpose has gained in recent years not only in the design of WDNs but also in other vast areas of applied GA. Messy GA with their variable string length feature and use of operators like cut and splice have lead to faster convergence in solutions. Based on the concept that global best solution will be found near the neighborhood of the region of relatively higher proportion of good solutions, the messy GA have proved very effective. Hybrid GA gives us the unique capability of using the best of both the local and the global search optimizers for the purpose of converging to the best global solution. The hybrid GA is useful in optimizing water quality and cost. Ant system algorithms have taken ideas from ants' foraging activity and have been shown to be very useful in the design of WDNs. The ant system algorithms make changes in the environment and not in the trial solutions itself. This makes it useful to tackle dynamically changing environments. This permits the researcher to model occurrences like real time pipe breakage or pump failure much easier. The ant algorithms used here are especially designed for discrete optimization. Shuffled complex algorithms though have proved to be efficient, yet they have not been used that extensively.

6. ACKNOWLEDGMENT

Would like to thank Dr. Ivan Garibay for his excellent advising and teaching, without which the understanding of the EAs would have been a tougher road.

7. REFERENCES

[1] Savic, D. A. and Walters, G. A. *Genetic Algorithms for the Least-Cost Design of Water Distribution Networks*. Journal of Water Resources Planning and Management, Vol. 123, No. 2, pp. 67-77, 1997.

[2] Bhave, P. R. *Optimal Design of Water Distribution Network*. Tata – McGraw Hill, 2003.

[3] Lansey, K. *Uncertainty in Water Distribution Network Modeling*. Journal of Contemporary Water Research and Education, Universities Council on Water Resources, Issue No. 103, 1996.

[4] Todini, E. *Looped water distribution networks design using a resilience index based heuristic approach*. Urban Water, Vol. 2, No. 2, pp. 115-122, 2000.

[5] Prasad, T. D. and Park, N. S. *Multiobjective Genetic Algorithms for Design of Water Distribution Networks*. Journal of Water Resources Planning and Management, Vol. 130, No. 1, pp. 73-82, 2004.

[6] Simpson, A. R., Dandy, G. C. and Murphy, L. J. *Genetic Algorithms Compared to Other Techniques of Pipe Optimization*. Journal of Water Resources Planning and Management, Vol. 120, No. 4, pp. 423-443, 1994.

[7] Goldberg, D. E. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison – Wesley Publishing Company, Inc., 1989.

[8] Gupta, I., Gupta, A. and Khanna, P. *Genetic algorithm for optimization of water distribution systems*. Environmental Modeling & Software, Vol. 14, pp. 437-446, 1999.

[9] Wu, Z. Y. and Simpson, A. R. *Competent Genetic-Evolutionary Optimization of Water Distribution Systems*. Journal of Computing in Civil Engineering, Vol. 15, No. 2, pp. 89-101, 2001.

[10] Tolson, B. A., Maier, H. R., Simpson, A. R. and Lence, B. J. *Genetic Algorithms for Reliability-Based Optimization of Water Distribution Systems*. Journal of Water Resources Planning and Management, Vol. 130, No. 1, pp. 63-72, 2004.

[11] Goldberg, D. E. *Computer-Aided Pipeline Operation using Genetic Algorithms and Rule Learning. PART I: Genetic Algorithm in Pipeline Optimization*. Engineering with Computer, Vol. 3. No. 1, pp. 35-45, 1987.

[12] Vairavamoorthy, K. and Ali, M. *Optimal Design of Water Distribution Systems Using Genetic Algorithms*. Computer-Aided Civil and Infrastructure Engineering, Vol. 15, pp. 374-382, 2000.

[13] Goldberg, D. E. *Real-Coded Genetic Algorithms, Virtual Alphabets and Blocking*. Complex Systems, Vol. 5, 1991.

[14] Beasley, D., Bull, D. R. and Martin, R. R. *An overview of genetic algorithms: 2. Research topics*. University Computing, Vol. 15, No. 4, 1993.

[15] Farmani, R., Walters, G. A. and Savic, D. A. *Trade-off between Total Cost and Reliability of Anytown Water Distribution Network*. Journal of Water Resources Planning and Management, Vol. 131, No. 3, pp. 161-171, 2005.

[16] Zitzler, E., Deb, K. and Thiele, L. *Comparison of Multiobjective Evolutionary Algorithms*. Evolutionary Computation, Vol. 8, No. 2, pp. 173-195, 2000.

[17] Deb, K., Pratap, A., Agarwal, S. and Meyarivan, T. *A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II*. IEEE Transactions on Evolutionary Computation, Vol. 6, No. 2, 2002.

[18] Formiga, K. T. M., Chaudhry, F. H., Cheung, P. B. and Reis, L. F. R. *Optimal Design of Water Distribution System by Multiobjective Evolutionary Methods*. Proceedings of the Evolutionary/Multi-Criterion Optimization: Second International Conference, Faro, Portugal, 2003.

[19] Farmani, R., Savic, D. A. and Walters, G. A. *Evolutionary multi-objective optimization in water distribution network design*. Engineering Optimization, Vol. 37, No. 2, pp. 167-183, 2005.

[20] Savic, D. A., Walters, G. A. and Schwab, M. *Multiobjective Genetic Algorithms for Pump Scheduling in Water Supply*. Selected Papers from AISB Workshop on Evolutionary Computing, pp. 227-236, 1997.

[21] Baran, B., Lucken, C. V. and Sotelo, A. *Multi-objective pump scheduling optimization using evolutionary strategies*. Advances in Engineering Software, Vol. 36, No. 1, pp. 39-47, 2005.

[22] Zyl, J. E. V., Savic, D. A. and Walters, G. A. *Operational Optimization of Water Distribution Systems Using a Hybrid Genetic Algorithm*. Journal of Water Resources Planning and Management, Vol. 130, No. 2, pp. 160-170, 2004.

[23] Tu, M. Y., Tsai, F. T. C. and Yeh, W. W. G. *Optimization of Water Distribution and Water Quality by Hybrid Genetic Algorithm*. Journal of Water Resources Planning and Management, Vol. 131, No. 6, pp. 431-440, 2005.

[24] Keedwell, E. and Khu, S. T. *A hybrid genetic algorithm for the design of water distribution networks*. Engineering Applications of Artificial Intelligence, Vol. 18, pp. 461-472, 2005.

[25] Keedwell, E. and Khu, S. T. *Novel Cellular Automata Approach to Optimal Water Distribution Network Design*. Journal of Computing in Civil Engineering, Vol. 20, No. 1, pp. 49-56, 2006.

[26] Keedwell, E. and Khu, S. T. *A novel evolutionary meta-heuristic for the multi-objective optimization of real-world water distribution networks*. Engineering Optimization, Vol. 38, No. 3, pp. 319-336, 2006.

[27] Halhal, D., Walters, G. A., Ouazar, D. and Savic, D. A. *Water Network Rehabilitation with Structured Messy Genetic Algorithm*. Journal of Water Resources Planning and Management, Vol. 123, No. 3, pp. 137-146, 1997.

[28] Halhal, D., Walters, G. A. and Ouazar, D. *Structured Messy Genetic Algorithm Approach for the Optimal Improvement of the Water Distribution Systems*.

[29] Dorigo, M. and Caro, G. D. *Ant Algorithms for discrete optimization*. Artificial Life, Vol. 5, pp. 137-172, 1999.

[30] Maier, H. R., Simpson, A. R., Zecchin, A. C., Foong, W, K., Phang, K. Y., Seah, H. Y. and Tam, C. L. *Ant Colony Optimization for Design of Water Distribution Systems*. Journal of Water Resources Planning and Management, Vol. 129, No. 3, pp. 200-209, 2003.

[31] Maier, H. R., Simpson, A. R., Zecchin, A. C., Leonard, M. and Nixon, J. B. *Ant Colony Optimization Applied to Water Distribution System Design: Comparative Study of Five Algorithms*. Journal of Water Resources Planning and Management, Vol. 133, No. 1, pp. 87-92, 2007.

[32] Zecchin, A. C., Simpson, A. R., Maier, H. R. and Nixon, J. B. *Parametric Study for an Ant Algorithm Applied to Water Distribution System Optimization*. IEEE Transactions on Evolutionary Computation, Vol. 9, No. 2, pp. 175-191.

[33] Zecchin, A. C., Maier, H. R., Simpson, A. R., Berrisford, M. J. and Leonard, M. *Max-Min Ant System Applied to Water Distribution System Optimization*. Online Proceedings of MODSIM 2003 International Congress on Modeling and Simulation, 2003.

[34] Liong, S. Y. and Atiquzzaman, M. *Optimal Design of Water Distribution Network Using Shuffled Complex Evolution*. Journal of the Institution of Engineers, Singapore, Vol. 34, No. 1, 2004.

[35] Eusuff, M. M. and Lansey, K. E. *Optimization of Water Distribution Network Design Using the Shuffled Frog Leaping Algorithm*. Journal of Water Resources Planning and Management, Vol. 129, No. 3, pp. 210-225, 2003.