University of Nevada, Reno

Evolving Potential Fields to Direct Tactics in Real Time Strategy Games

A thesis submitted in partial fulfillment of the
requirements for the degree of Master of Science in Computer Science

by

Michael Oberberger

Dr. Sushil Louis, Thesis Adviser

December 2010

THE GRADUATE SCHOOL

We recommend that the thesis
prepared under our supervision by

MICHAEL OBERBERGER

entitled

Evolving Potential Fields To Direct Tactics In Real Time Strategy Games

be accepted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE

Sushil Louis, Ph.D., Advisor

Konstantinos Bekris, Ph.D., Committee Member

Donald Pfaff, Ph.D., Graduate School Representative

Marsha H. Read, Ph. D., Associate Dean, Graduate School

December, 2010

Abstract

This thesis investigates the use of co-evolution to generate tactics in real-time strategy games. Games like Chess have been used to test AI approaches since the 1960s. Modern video games with simulated worlds now allow us to investigate AI approaches in less abstract spaces, thus allowing research results to be perhaps more immediately applicable. Real-time strategy games, a genre of modern video games, are widely used in wargaming and what-if scenario analysis in the military and in industry. Since good tactics can determine whether you win or lose, this thesis focuses on competent tactics generation in real-time strategy games.

There are many ways to generate players for games. Many classical approaches employ a system of logic that relies on expert knowledge about the game and perhaps known effective strategies. Although they are good for certain kinds of problems, expert systems' approaches have been shown to be brittle and generally do not learn from experience. This work uses an evolutionary approach to learning tactics for RTS games. Since evolutionary techniques are generally good at learning solutions to specific problems, this work also employs co-evolutionary techniques to generate more robust tactics that are effective against potentially unseen opponents.

My results from generating tactics against a specific opponent indicate that an evolutionary algorithm can evolve good tactics. The generated tactics defeat a known opponent after a relatively short training cycle. However, these tactics are specific to the opponents that were trained against - they do not perform as well against other opponents.

Co-evolution leads to more adaptive tactics. My results from employing a co-evolutionary approach indicate that co-evolution can generate tactics that perform better over a set of previously unseen opponents.

These results indicate the potential for co-evolutionary and evolutionary approaches to tactic generation. Because they may not be biased by human preconception, I believe that such approaches also have the potential to generate completely new and surprising tactical solutions to difficult problems.

Acknowledgments

I would like to thank my committee members Dr. Konstantinos Bekris and Dr. Donald Pfaff for their valuable time and feedback.

Without the help of my parents, Walter and Marlies, I would not have had the time to get this work completed. They have helped me enormously during this process. I cannot thank them enough.

Dr. Sushil Louis has dedicated a lot of his time to helping me get this work to a presentable state. I appreciate his aid, insight, and monumental patience. He has my most humble gratitude for his efforts.

Acknowledgment goes to the Office of Naval Research for providing me funding during this research under award number N00014-09-1-1121.

Finally, to my twin sons Daeven and Cole, I cannot ever express how thankful and fortunate I am to have you in my life. You have made me a better person by letting me be your Daddy. Without you guys, I would still be working for a corporation somewhere and not living a life that makes me happy. I will always be beholden to you for how you've affected my life's ambitions.

Contents

Listings

List of Figures

Chapter 1

Introduction

Generating tactics to implement strategies in RTS games is important when one is interested in providing interesting opponents which are controlled by computers. There are many ways to implement these tactics in computer programs. Typically, computer programs have implemented logic that included the knowledge and experience of human beings. This has led to Artificial Intelligence (AI) opponents which could be described as predictable. It would be interesting to provide new tactics that were less predictable, yet still effective.

This thesis focuses on using evolutionary algorithms to generate new tactics that are not based on any pre-conceived notions about how to play a game. I show that it is possible to evolve tactics that start with no tactical knowledge of the game, yet are still able to defeat opponents with regularity.

I focus on tactics in a game with one goal - eliminate the opponent. The tactics are evolved to accomplish this. Namely, it is not the goal to merely survive in the game - one must eliminate the other player to win.

There is a well-known problem in machine learning: over-specialization. I find that the tactics that evolve over-specialize against the opponents used during evolution. In the second part of my work, I address this challenge by implementing a co-evolutionary algorithm that tends to generate more robust tactics. I show that these co-evolved tactics are better suited to defeat opponents that were never encountered during evolution than those tactics generated by evolution alone against specific opponents.

1.1 Evolving Tactics in Games

There are many known methodologies for defining a tactic for AI agents in Real-Time Strategy (RTS) games [Hagelback and Johansson, 2008, Avery, et.al. 2009, Leigh, et.al. 2008]. Many of these tactics

rely on some scripted, or hard-coded, behavior. This behavior is typically dependent on the ability to control and coordinate all of the actions of all movable objects, or vehicles, in the game. The AI is assumed to have knowledge and a certain level of control over the state of each vehicle.

In some of the approaches cited above, the game environment uses expert logic to implement tactics. Examples of this logic are:

- If the vehicle is close to another vehicle, then move away from that vehicle.
- Find the vehicle with the most strategic value and target it
- Group all vehicles such that they create a predefined formation
- If a vehicle has taken damage, do not engage the enemy

These are only a small subset of the rules that may apply to a tactic. They serve only as examples of expert logic.

The topic of evolutionary algorithms has been studied since the 1960s [Fogel and Owens, 1966]. They have been particularly successful at searching high-dimensional spaces for solutions to a variety of problems. I show that using evolutionary algorithms to tune the parameters used in determining the strengths of *potential fields* is successful at producing tactics that are effective against a range of opponent tactics.

My research shows that effective tactics can be evolved using only potential fields. These fields act to either attract or repulse one vehicle from another. By tuning the strength of these fields in the correct manner, I show that these tactics can be effective in neutralizing an opponent player.

1.2 Thesis Structure

Chapter 2 explains the necessary background material and topics used in this research. I include overviews of evolutionary algorithms, real-time strategy games, and potential fields. Then, past work on these topics is explained.

Chapter 3 covers the methodologies used to evolve tactics using potential fields. I describe the RTS game designed for use in this thesis. Then, the encoding of tactics in a manner conducive to evolution

is explained. I describe my evolutionary algorithm, along with how it differs and how it is similar to other evolutionary algorithms. Finally, my co-evolutionary algorithm is explained.

Chapter 4 describes the specifics and results of the experiments conducted in this research. I include commentary on how the evolutionary process worked in the experiments. Then, I show that it is possible to derive effective tactics using only potential fields when using evolutionary and co-evolutionary algorithms.

Chapter 5 discusses the conclusion of the experiments. I also include some of the future work that this thesis could inspire.

Chapter 2

Background

This chapter describes RTS games, virtual potential fields, and evolutionary algorithms. The primary focus of my research is in evolutionary algorithms, so those are described first. The application of evolutionary algorithms to my problem is next. Finally, I outline other related work pertaining to the use of potential fields in games and the use of evolutionary algorithms in games.

2.1 Evolutionary Algorithms

Evolutionary algorithms have been studied for decades [Fogel and Owens, 1966, Fogel and Burgin, 1969]. They were formalized by John Holland in the 1970's [Holland, 1975]. The Genetic Algorithm (GA) in Holland's work was based on a simplified view of the evolution of a biological system according to certain Darwinian principles. GAs have been found to be quite effective in searching high-dimensional multi-modal solution spaces, especially when the function describing the *fitness* of a solution is a non-continuous function.

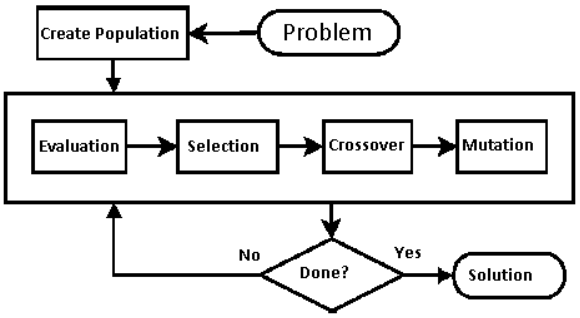
2.1.1 The Canonical GA

The canonical GA starts with a problem and a randomly created population of possible solutions, referred to as *chromosomes* to mirror their biological counterpart. Each chromosome contains data that, once decoded appropriately, is proposed as a candidate solution to the problem. The decoded data's ability to solve the problem becomes that chromosome's *fitness*.

The GA evaluates all of the chromosomes' fitnesses at the start of every *generation*. It then uses three operators- selection, crossover, and mutation- to create the next generation of chromosomes. Figure 2.1 shows the process used by the canonical GA.

Selection chooses chromosomes to act as parents when producing offspring. It biases towards higher

Figure 2.1: The Canonical GA Algorithm



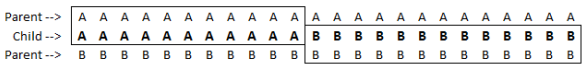
fitness chromosomes, assuming that high fitness parents are more likely to produce high fitness offspring. The canonical form of selection is the *roulette wheel* method, where a chromosome's probability of selection is proportional to its fitness. Therefore, higher fitness chromosomes receive a better chance of becoming parents than lower fitness chromosomes.

Each selection of a parent happens independently of previous selections. This means that chromosomes with higher fitness can be chosen multiple times and can create many offspring with multiple partners.

After the couples are selected, they are, with a certain probability, affected by the crossover operator. Crossover combines parts from two parents to make two offspring. If the parts of each parent that lead to high fitness get combined in the right manner, the offspring can potentially have higher fitness than their parents. In turn, high fitness offspring have a good chance of becoming parents in the next generation, which further pushes the population towards better solutions.

The canonical method of crossover is *single-point crossover*. In this method, the crossover operator chooses the same random point in both parents. The first part of the first parent and the second part of the second parent form a first new offspring, while the opposite parts form a second new offspring. See Figure 2.2.

Figure 2.2: Single Point Crossover



After crossover of couples creates the new population, that population is subject to the mutation operator. Mutation has a small chance of randomly changing some part of a chromosome. Mutation

is insurance against permanent loss of genetic material [Sivanandam and Deepa, 2008].

2.1.2 Evolutionary Algorithms in General

In evolutionary systems, one is typically interested in maximizing (or minimizing) some function $f(x_1, x_2, ...x_n)$ where each parameter x_i is limited by some "alphabet". In Holland's work, this alphabet is the binary alphabet: $x_i \in \{0, 1\}$. This is the alphabet used for the *canonical genetic algorithm*- the algorithm most studied by researchers in this field [Holland, 1975].

There are many functions f that are not easily defined by, or are not well suited for, a binary alphabet. In Antonisse's work, an argument is presented that, with a different interpretation of schemata, there are many more schemata using non-binary alphabets than with binary alphabets [Antonisse, 1989]. Wright shows that it is possible to present an alphabet of real values and, with certain crossover operators, Holland's schema theorem holds true [Wright, 1991]. My research builds on that of Wright, in that it presents a real-valued alphabet for the vector of parameters passed to function f.

In a canonical GA, the use of a *generation* provides one type of approach to evolution. Each generation involves the operations mentioned to evolve a completely new population of chromosomes. However, other work, including Whitley's, abandons the generational approach in favor of creating new chromosomes on an individual basis [Whitley, 1989]. This *Genitor* approach is one of the bases of my research.

2.2 Potential Fields

Potential fields are omnipresent in our universe. Two of the most commonly recognized potential fields are seen in models of magnetics and gravity. While advances in quantum theory may dispute these as proper forces of attraction[1], the observable model of both gravity and magnetics is still a natural and pervasive idea.

Gravity's potential field between two bodies of mass is typically described using the equation: $F_G = \frac{gm_1m_2}{d^2}$. This equation can be generalized as $F = cd^e$, where c is a real coefficient, d is the

[1] Gravity, for instance, is now typically explained using a field model, where masses distort spacetime rather than actually attract each other

distance between two objects, and e is a real exponent. In the case of the simplified equation of gravity, $c = gm_1m_2$ and $e = -2$. Using this general equation $F = cd^e$, we can define a potential field between vehicles which is useful in providing a tactic.

2.3 RTS Games

There are many types of RTS games available today. These RTS games are generally focused on eliminating your opponent. However, in order to accomplish this, there are commonly many sub-goals in the game. These subgoals include resource collection, creation of buildings, creation of combat and utility units (based on what buildings have been built), etc. This work does not concentrate on any of these sub-goals of RTS games. It focuses specifically on providing tactics for the movement of vehicles which maximizes the ability to eliminate the other player, given the vehicles and game rules available.

2.4 Other uses of Potential Fields and Evolution in RTS Games

There has been other research in using potential fields or evolution to provide strategies in RTS games. For instance, in Leigh's work, a canonical GA is used to evolve a specific set of parameters selected by the authors based on what they deemed to be relevant to the game [Leigh, et.al. 2008]. Some of these parameters included:

- When is a target considered close
- What is the range of vision
- What field of view should be used to recognize targets

Each of these parameters worked in conjunction with a piece of program logic to alter the behavior of a unit in some way. Overall, there were approximately 22 parameters that were evolved. Each of the parameters was approximated using a binary alphabet, where the precision of each parameter was based on the author's judgment of what would be appropriate.

In Hagelback and Johansson's work, the authors used potential fields to drive all tactics in a full RTS game [Hagelback and Johansson, 2008]. This includes resource gathering, unit production, exploration, etc. There were approximately 10 different types of fields defined in the research. The resulting

computer-controlled agent did well in an international RTS tournament, defeating the previous year's winner 330-70 over 400 games. However, all of the potential fields employed by this research had parameters (coefficients, exponents, etc) that were hand-selected by the authors. Many of the fields employed were discrete functions, involving *if-then* type logic to determine what form the field would take. They did not describe any machine learning methodology to optimize the parameters to any of their potential fields.

In Avery's work, *influence maps* were used to derive tactics for ships in a *capture-the-flag* setting [Avery, et.al. 2009]. In defining the influence maps, this work used a form of potential field to derive the *influence* for each of a set of grid cells. The battlefield was represented by a grid of discrete cells. Each cell may be empty, contain a friendly or enemy unit, or contain the flag. Avery's work used a GA to evolve the parameters used by the potential field when generating the influence map. However, once the influence map was generated, it used a path planner to direct the movements of the vehicles based on the influences in each grid cell.

In my work, I show that using a combination of potential fields and evolutionary algorithms, one can define and generate effective tactics. Unlike Leigh's work, I do not have any expert logic that defines part of the tactic. Unlike Hagelback's work, I do not hand-craft the parameters to the potential fields. And unlike Avery's work, I use the potential fields in an online manner to direct the movement of vehicles, rather than use a path planner.

2.5 Summary

In this chapter, I covered some background information on evolutionary algorithms, RTS games, and potential fields. In the next chapter, I explain the methods and algorithms I used to evolve the strengths of potential fields such that they provided effective tactics in an RTS game.

Chapter 3

Methodology

I begin this chapter by describing a simple RTS game that provides the rules and fitness function for my evolutionary algorithm. Next, I explain how the potential fields were designed such that they were able to provide input controls for the game. I then describe how the chromosomes for the evolutionary algorithm were encoded such that they provided a representation of the potential fields. Following that, I explain how my basic evolutionary algorithm operates. Finally, I show how co-evolution was implemented.

3.1 The RTS Game

The first step in this research was providing an infrastructure in which to play an RTS game. The game was designed such that it balanced realism with simplicity. The basic game design envisioned a 2-player game where each player controls N vehicles of identical types. Each unit has basic movement constraints, similar to those found in popular RTS games such as Starcraft and Warcraft. Each vehicle can fire upon opponent vehicles, provided the opponent is within range. The game is over only when all of one player's vehicles are destroyed, presumably by the other player. For this work, an open playing field was used. No obstacles were introduced.

A few assumptions were made about the way the game is played.

- Because the goal of the game is to destroy the opponent, a vehicle will always fire upon an opponent vehicle if it is able to. The game enforces this.

- Control of movement of vehicles is a core component of the tactics that were being learned. Therefore, the game imposes no restrictions on the collision of vehicles. In a typical RTS game, the game engine itself will minimize the effect of collisions. It will generally either allow the

vehicles to overlap each other, or it will allow the vehicles to touch, but not move into each others space. Generally, an RTS game will not damage vehicles which collide, while in my game collisions are something that can severely penalize a player.

3.1.1 Vehicle Information

Each vehicle in the game has the following statistics:

Statistic	Typical Value	Purpose
Hitpoints	100 hps	The basic unit of health for a vehicle. A vehicle's Hitpoints is decremented when (a) that vehicle is hit by an opponent's weapon; and (b) when a vehicle collides with any other vehicle. A vehicle is destroyed when its Hitpoints reaches zero.
Size	20 units	The size of the vehicle. Used to determine if two vehicles have collided. Also used to determine how close one vehicle is to another. A *unit* is an arbitrary unit of length, currently set as one pixel on the computer screen.
Maximum Speed	25 units / second	This is the maximum speed of a vehicle, in terms of units per second.
Maximum Reverse Speed	-10 units / second	This is the maximum speed of a vehicle when it is moving in the opposite direction of its current orientation in the game
Seconds to Max Speed	2 seconds	The amount of game time required to alter the speed of the vehicle from a complete stop to its *Maximum Speed*. This is an approximation of acceleration.
Seconds to Full Circle	5 seconds	The amount of game time required for the vehicle to turn in a complete circle.
Max Damage	20 hps	The maximum number of hitpoints that are removed from the target when this vehicle fires at another vehicle. This value is decreased in linear proportion to the distance between the two vehicles to approximate the chance to miss the target.
Range	100 units	The maximum distance at which the vehicle can fire at another vehicle. If a target vehicle is farther than *Range*, a vehicle cannot fire upon that target.
Armor	10 hps	The amount of damage that is absorbed by the armor of the vehicle. This value is removed from the incoming damage from weapons fire before the damage is applied to (removed from) this vehicle's hitpoints.
Hit Chance at Max Range	50%	When at maximum range, this value is multiplied by the *Max Damage* of the vehicle before the shot hits the target. This approximates the chance that the vehicle has of missing the target based on how far away it is from its target.
Cooldown	1 second	The amount of game time that must elapse between shots.

3.1.2 Vehicle Collisions

When two vehicles collide (regardless of which player controls the vehicles), each vehicle's current hitpoints are subtracted from the other vehicle's current hitpoints. The net effect of this calculation is that one of the vehicles will be destroyed while the other vehicle will be damaged. In the case that

both colliding vehicles have exactly the same number of hitpoints, both will be destroyed.

In a real-world scenario, say for instance ships at sea, if two ships were to collide there would be consequences. Some damage will be done when ships collide. This damage can be viewed from two points of view. On the one hand, damage is generally not considered a good thing. On the other hand, if one player chooses to be suicidal, ramming an opponent's ship may be in the overall best interest of the player. I decided to enforce these rules around collisions as part of the game mechanics and let the evolutionary algorithm decide whether or not collisions were in the best interest of the AI.

3.1.3 Weapons Fire

When a vehicle has multiple targets (opponent vehicles) within its range, it will fire upon the opponent vehicle with the fewest remaining hitpoints. While this is not the optimal method for designing a firing solution, its simplicity is appropriate for these simulations.

3.1.4 Movement of Vehicles

Each vehicle is controlled by two parameters. These are *desired speed*, a scalar value in terms of units per second, and *desired heading*, another scalar value in terms of radians.

The game uses the characteristics of the vehicle, namely *Seconds to Max Speed* and *Seconds to Full Circle*, combined with the current state of the vehicle (*current velocity*, which is used to derive *current speed* and *current heading*), to alter the speed and heading of each vehicle.

There is a movement heuristic provided by the game- If $ABS(desiredHeading - currentHeading) > \frac{\pi}{2}$, the game will reverse the desired speed until this condition is no longer met. This has the effect of slowing down vehicles when their desired heading is behind them. This allows vehicles to turn around more quickly.

3.1.5 Score

A player loses the game once all of its vehicles are destroyed. The score for a player is calculated using the following formula:

$$Score_{Winner} = \left(\frac{RHP_{Winner} \cdot RVC_{Winner}}{HP_{Winner} \cdot VC_{Winner}} \right)^{\frac{\left(\frac{HP_{Winner}}{HP_{Loser}}\right)^2}{2}}$$

$$Score_{Loser} = -Score_{Winner}$$

where

HP**:** The sum of the starting hitpoints for the player.

RHP**:** The sum of the remaining hitpoints for all vehicles not destroyed at the end of the game. This value is zero for the loser of a game.

VC**:** The total number of vehicles for a player.

RVC**:** The count of remaining vehicles (vehicles that were not destroyed) for a player at the end of the game.

The range of this function for either player is $[-1, 1]$. In the case of a tie, which occurs when both players' final shots destroy the other player simultaneously, the score is zero. This formula gives credit to the winner for its total remaining hitpoints and its total remaining vehicle count. Further, it is meant to take into account any differences in team sizes - defeating an opponent that has more total hitpoints will yield a better score than defeating an evenly matched opponent. I consider the score of a game[1] as the fitness function for the evolutionary algorithm.

Based on the parameters to this function, it should be obvious that *remaining hitpoints* and the percentage of vehicles remaining play an important part in generating a good score. By introducing collision damage the way that I did, I provide a *fitness landscape* that is sharply affected by any collisions. This can provide a severe penalty when vehicles collide, especially if the two colliding vehicles are on the same team. The implications of this effect is reviewed later in this thesis.

[1] In most training scenarios, the average of a set of scores

3.1.6 The Agent

It should be obvious that there are numerous methods available for programming an AI to play this game. The most basic AI will set the desired speed and the desired heading to some constant value. Since the game will fire weapons automatically, this simple agent will be able to inflict *some* damage on any opponent that engages it.

For this work, an AI lovingly named *Herkermer* was designed. Herkermer uses the potential fields generated by all vehicles in the game to provide the controls for the movement of its vehicles (desired speed and heading). Using the positions of every remaining vehicle in the game, Herkermer can obviously calculate the distances between the vehicles. These distances, combined with the data found in the chromosome that configures Herkermer, are used for the potential field calculations. The results of the potential field calculations are then used as feedback into the controls for the vehicle.

3.2 Potential Fields

Potential fields between vehicles are what drive the movement controls for each vehicle under the control of Herkermer. Herkermer calculates the desired speed and heading by first summing the potential field vectors, then by breaking down the sum into its component parts (magnitude and orientation). How Herkermer uses the information found in the chromosome to navigate the vehicle is described below.

3.2.1 Potential Field Composition

A potential field is comprised of three *virtual* forces. Each of these virtual forces is described by the simple equation

$$F = cd^e$$

where

c: A coefficient

d: The distance between vehicles

e: An exponent

Intuitively, the distance is provided by the game state. The c and e values are provided by the chromosome and evolved during the evolutionary process.

The three *virtual forces* which form a potential field are:

Attractor: The attractor has a stronger effect on vehicles that are close to each other. It simulates the force of gravity, which increases in intensity relative to the inverse of the distance squared between two bodies. A typical attractor may look like $F = 2000d^{-2}$.

Repulsor: The repulsor acts as a sort of "negative gravity", or similar to opposing poles of magnets. It is used to keep vehicles from getting too close to each other, thereby avoiding collisions. It is typically stronger than the attractor, but with a higher (negative) exponent. This allows the attractor to bring vehicles closer together until they reach the point where the repulsor provides an equal opposing virtual force. A typical repulsor may look like $F = 500000d^{-3}$.

Spring: The spring is used to bring vehicles that are far away from each other closer together. When vehicles are far apart, the magnitude of the attractor alone is relatively small (and the repulsor smaller still). The spring's value is higher when vehicles are farther away, thus drawing vehicles into the conflict from wherever they may be on the playing field. A typical spring may look like $F = 1.0d^{0.9}$.

Each vehicle in the game (that has not been destroyed) produces a potential field comprised of these three virtual forces. Each potential field is described by 6 parameters, which are the coefficient and exponent for each of the three forces:

$$PF = \{A_c, A_e, R_c, R_e, S_c, S_e\} = \{x_0, x_1, x_2, x_3, x_4, x_5\} \tag{3.1}$$

Herkermer uses a set of two potential fields (PFS) to determine the effect of another vehicle on any vehicle it controls:

$$PFS = \{PF^0, PF^1\}$$

The potential field PF^0 contains the field parameters used when the vehicle under control has no hitpoints (explained shortly). The potential field PF^1 denotes the parameters used when the vehicle

is at full hitpoints. A linear interpolation between corresponding parameters in the two fields defines the effective parameters used based on the current health of the vehicle. This is the *effective potential field* (EPF), and it is calculated based on PFS and the remaining hitpoint percentage:

$$EPF = lerp(PF^0, PF^1, hp\%)$$

where $hp\%$ is the percentage of remaining hitpoints for the vehicle under control and $lerp$ is a linear interpolation function. For instance, if the vehicle under control is at 50% health, EPF will be calculated as:

$$EPF = \{\frac{x_0^0 + x_0^1}{2}, \frac{x_1^0 + x_1^1}{2}, ..., \frac{x_5^0 + x_5^1}{2}\}$$

where the x values are those from Equation (3.1) for each of PF^0 and PF^1 respectively.

3.2.2 Encoding

Herkermer's configuration chromosome requires a matrix, or grid, of potential field sets. This grid is composed of the PFS values between two types of units. For each type combination, two PFS vectors need to be provided- one for friendly units of that type, and one for enemy units of that type. For this research, only one type of vehicle was used, so the simple grid is shown in Figure 3.1.

Figure 3.1: Simple Grid for One Vehicle Type

PFGrid	PFS values
Friend	PFS^F
Enemy	PFS^E

However, for a game with two vehicle types, the matrix would look like Figure 3.2.

Figure 3.2: Grid for Two Vehicle Types

PFGrid		Type 0	Type 1
Type 0	Friend	PFS_{00}^F	PFS_{01}^F
Type 0	Enemy	PFS_{00}^E	PFS_{01}^E
Type 1	Friend	PFS_{10}^F	PFS_{11}^F
Type 1	Enemy	PFS_{10}^E	PFS_{11}^E

The number of PFS vectors ($PFS\,Count$) grows exponentially with the number of types N:

$$PFS\,count\ =\ 2N^2$$

This collection of PFS values is called a $PFGrid$. The $PFGrid$ serves as the configuration input for Herkermer. Herkermer gets all other information (vehicle positions, etc) from the game itself once the game starts.

The $PFGrid$ also serves as the DNA for the *chromosome* in the evolutionary algorithm. For the current work, with only one vehicle type, this chromosome is made up of $6 * 2 * 2 = 24$ real values. Stated differently, the chromosome represents a vector in 24-dimensional space.

3.2.3 Spatial Diversity and Distance

Since the components of the $PFGrid$ represent strengths of potential fields, the difference in these components can represent significantly different tactics once decoded. For instance, if the strength of the repulsor is changed by a significant amount[2], two vehicles that would otherwise not collide may now collide. This made me interested in defining a *distance metric* between two $PFGrid$s.

A $PFGrid$ represents a vector, or a point, in N-dimensional space. N is at least 24 in this work. To quote Aggarwal, "*Recent research results show that in high dimensional space, the concept of proximity, distance, or nearest neighbor, may not even be qualitatively meaningful*" [Aggarwal, et.al. 2001]. In my experiments, I found that using the Euclidean Distance ($\sqrt{\sum_i (x_i^1 - x_i^2)^2}$) yielded similar numbers regardless of how different the tactics represented by the $PFGrid$s were. Eschelman uses the Hamming distance to determine whether or not two chromosomes are *close* in the search space [Eschelman, 1990].

Knowing something about the search space, namely that two tactics can behave differently if any one of their $PFGrid$ components are significantly different, I chose to use what I call a *Floating Point Hamming Distance* when calculating whether or not two $PFGrid$s were close in the search space.

After some of my initial work was completed, but before I had any meaningful results, I had a sample database of roughly 40,000 $PFGrid$s. I took a "snapshot" of these grids and created a vector of values based on the standard deviation for each value x_i from Equation (3.1). I called this vector the *Standard Deviation Vector* (STV). I use it in calculating my *Floating Point Hamming Distance*.

The calculation is straightforward:

$$Distance(G^0, G^1) = \sum_{i=0}^{23} Int\left(\frac{|G_i^0 - G_i^1|}{SDV_{i \bmod 6}}\right) \tag{3.2}$$

[2]What is considered *significant* is what the evolutionary algorithm is meant to determine

where G^0, G^1 were two *PFGrid*s, *SDV* was the 6-dimensional *Standard Deviation Vector*, *Int* is a function that returns the integer value of a real number, and *mod* is the modulo operator. In this thesis, *distance* or *spatial diversity* refers to Equation (3.2).

3.3 Evolution

I designed an algorithm called Diversity Preserving Evolutionary Algorithm (DPEA). I wanted an algorithm that would both advance the fitness of a population of chromosomes as well as maintain a spatially diverse population of chromosomes simultaneously. Deb and Goldberg provided mechanisms for accomplishing this in their work, but this work concentrated on binary alphabets, not real alphabets [Deb and Goldberg, 1989]. By maintaining a higher diversity in the chromosomes, I was trying to avoid getting "stuck" on local maxima in the solution space. DPEA was shown to be effective in accomplishing this goal.

As with most every evolutionary algorithm, I start with a population of randomly generated *PFGrids*, or chromosomes. For most of the trials, this population was 100 chromosomes. This initial set of chromosomes comprises *epoch 0*. The size of the initial population (usually 100) is called the *initial epoch size*, or *IES* for the population.

An *epoch* is something that I defined to provide a reset point for the population during evolution. As the population's overall fitness increases, a new epoch is reached. A new epoch is designed to trim down the population, leaving a more concise set of valuable chromosomes. This process is explained below.

Similar to the way Whitley defines the Genitor algorithm [Whitley, 1989], I select one or two chromosomes from the current epoch of the population and perform some operation on these chromosomes. The operations available are versions of crossover and mutation, commonly found in most evolutionary algorithms. These operations are explained below. Using one of these operations, I create a new chromosome from one or two parents. This new chromosome is then evaluated and a new fitness is assigned. The chromosome is deemed *acceptable* according to the algorithm shown in 3.1.

The new chromosome is thrown away if any of the following conditions is true:

- While evaluating the chromosome, one of the games played didn't complete in the allocated

Listing 3.1: Is a new chromosome acceptable?

```
FUNCTION Is_Acceptable(pop,ch)
    IF NOT ch.Completed_All_Games
        RETURN FALSE

    IF MIN( Distance( ch, pop.AllChromosomesInEpoch )) = 0
        RETURN FALSE

    LET avg = AverageFitness( pop.AllChromosomesInEpoch )
    IF ch.Fitness < avg
        RETURN FALSE

    RETURN TRUE
END_FUNCTION
```

amount of time. In this case, the chromosome, when decoded, results in an invalid tactic.

- The chromosome is deemed spatially identical[3] to another chromosome in the population, but its fitness is worse than the existing chromosome. In this case, the population already contains a better version of the same chromosome.

- The chromosome's fitness is worse than the average fitness in the population. In this case, we didn't produce a very good chromosome.

Otherwise, the chromosome is added to the population. This set of operations is repeated until the current epoch is ready to become the next epoch.

Generation of more epochs continues until I decided to stop it. Normally, I stopped evolution when I saw that there was no progress being made in the overall fitness of the current epoch after thousands of evaluations of new chromosomes.

The pseudo-code for this algorithm is found in Listing 3.2.

3.3.1 Epochs

An epoch is designed to maintain spatial diversity among the best chromosomes in the population. The goal is to maintain a diverse set of chromosomes all of which have good fitness. This work was inspired by the notion of *niching* [Deb and Goldberg, 1989, Fonseca and Fleming, 1993], and is a key component of DPEA.

[3] Its *floating point hamming distance* is 0

Listing 3.2: High Level Pseudocode for Evolution

```
LET pop = Initial_Population

DO UNTIL Some_External_Termination_Event
    WHILE NOT Epoch_Is_Done(pop)
        LET ch = Generate_New_Chromosome(pop)
        IF IsAcceptable(pop, ch)
            Add_To_Epoch(pop, ch)
    END_WHILE

    Evolve_Epoch( pop )

END_DO
```

Every epoch begins with exactly IES chromosomes. In the case of epoch 0, these are randomly generated chromosomes. In the case of all other epochs, these initial IES chromosomes come from the previous epoch. When the average of the best IES chromosomes in the current epoch exceed the average of the best $\frac{IES}{4}$ chromosomes that initialized the epoch, a new epoch is generated. The pseudocode for determining when to move to the next epoch is in Listing 3.3.

Listing 3.3: Pseudocode for checking if an Epoch is done

```
FUNCTION Epoch_Is_Done( pop )
    LET initial = pop.InitialChromosomes.OrderByFitnessDescending
    LET target = AverageFitness( initial.Take( INITIAL_EPOCH_SIZE / 4 ) )
    LET best = pop.AllChromosomes.OrderByFitnessDescending
    LET current = AverageFitness( best.Take( INITIAL_EPOCH_SIZE ) )

    IF current > target
        return TRUE
    ELSE
        return FALSE

END_FUNCTION
```

Chromosomes will be continuously added to the population until the transition condition is met. As such, it is possible to have epochs with 200 chromosomes in them. It is also possible to have epochs with 5,000 chromosomes in them. The number of chromosomes in an epoch does not define an epoch. In the Discussion chapter, I hypothesize about the implication of a large chromosome count for an epoch.

I developed an algorithm for deciding which chromosomes from the current epoch will make it into

the next epoch. The goal of this algorithm was to move a set of chromosomes forward that had good fitness *and* good spatial diversity. In researching statistical clustering techniques in high dimensions [McCallum, et. al 2000, Eppstein, 1999], I found that this problem is quite a complex field of study unto its own. What I found was that all of the techniques available were computationally expensive. I wanted an algorithm that balanced good clustering with fast execution speeds. This led me to the algorithm outlined in Listing 3.4.

I borrowed from the techniques that are well known, while reducing the iterative nature of the existing algorithms. Essentially, I consider all of the chromosomes in the epoch with above average fitness. Starting at the best chromosome, I look for the set of all of the chromosomes that are far away spatially from this chromosome. As I add each chromosome to this set, the condition for the next chromosome is that it is far away from every chromosome that is currently in this set. This makes it more and more difficult to get added to this set. When there are no more chromosomes in the above average chromosomes that are far away from this set, I add the set to the next epoch and repeat, using the next-highest fitness chromosome that still remains. I continue to do this until I have IES chromosomes forwarded to the new epoch.

This gives the new epoch a starting set of chromosomes that are all above average in fitness and are also spatially diverse.

3.3.1.1 Testing Epoch Evolution

Since this algorithm is my own method for *niching* [Deb and Goldberg, 1989], I was interested in how it would perform against some well known multi-modal solution spaces. So I designed a test for this algorithm. Using the same model used for evolving chromosomes for the RTS tactics (a variation on Genitor, and described above), I ran my own simulations on one of the functions used in Deb and Goldberg's research: $F(x) = sin^6(5\pi x)$ where $0.0 \leq x \leq 1.0$. My algorithm for epoch evolution was able to find and maintain all 5 peaks provided by this function using a population size of 50 and a simple hill climber as a chromosome generator. Changing this function to a 25-peak function, $F_2(x) = sin^6(25\pi x)$, my algorithm was able to maintain all 25 peaks over 100 runs 95% of the time. For the 25 peak trials, I used a population size of 300 and again a simple hill-climber. Empirically,

Listing 3.4: Pseudocode for Evolving an Epoch

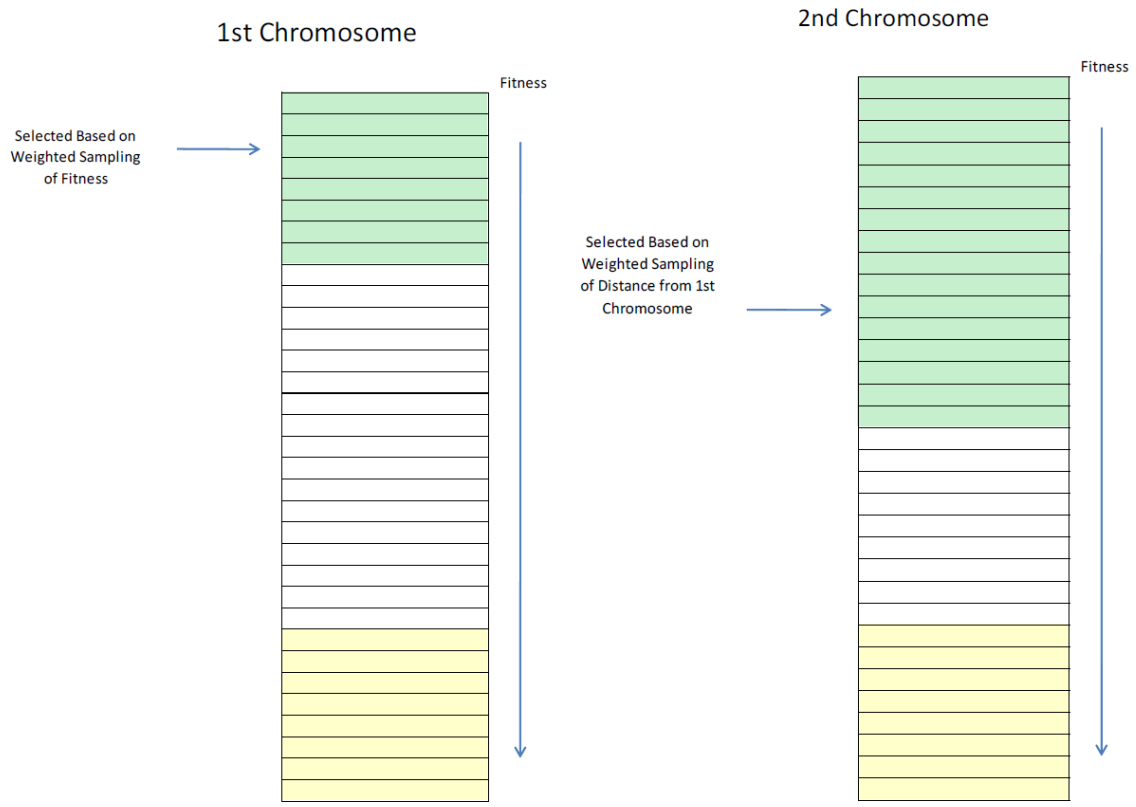
```
PROCEDURE Evolve_Epoch(pop)
    LET available = pop.AboveAverageChromosomesInEpoch
    LET selected = <empty>

    WHILE selected.Count < EPOCH_SIZE

        LET ch = available.First()
        LET working = { ch }
        available.Remove( ch )

        Increase_Working_Set( working, available )

        selected.Add( working )
    END_WHILE

    pop.Epoch = pop.Epoch + 1
    pop.ChromosomesInEpoch = selected

END_PROCEDURE

PROCEDURE Increase_Working_Set( working, available )
// "found" is TRUE as long as we are adding to the working set
    LET found = true
    WHILE found
        FOR_EACH ch IN available
            found = true
            FOR_EACH good IN working
                IF good.Distance(ch) < 1
                    found = false
            END_FOR_EACH
            IF found // ch is not close to any "working"
                working.Add( ch )
                available.Remove( ch )
            END_IF
        END_FOR_EACH
    END WHILE
END_PROCEDURE
```

this shows that my algorithm for epoch generation does indeed maintain multiple spatially diverse chromosomes that also have high fitness.

3.4 Chromosome Selection

In order to generate a new chromosome, one or two parent chromosomes must be selected from the chromosomes in the population. The number of parents is determined by the actual chromosome generator that is used. All generators use either one or two parents. They all use the following algorithms for the selection of their parent(s). Figure 3.3 shows this graphically.

Figure 3.3: Selection of Parent Chromosomes



Every chromosome generator requires at least one parent. This parent is selected from a set of *IES* chromosomes in the current epoch of the population with the best fitness values. I use simple *roulette wheel* selection based on fitness.

Some chromosome generators require two parents. The second parent chromosome, if one is re-

quired by the generator, is selected differently. Because we want to explore the solution space as much as possible, we want to select a second parent that is farther away spatially from the first parent but still has above-average fitness. The set of available chromosomes for the second parent is comprised of all of the chromosomes in the current epoch of the population that have a fitness value that is above average for that epoch. Again, a chromosome is selected using a simple *roulette wheel* selection, but this time the weights for this second selection are based on the distance of the chromosomes from the first parent.

3.5 Chromosome Generation

In the canonical GA, there are two main methods for new chromosome generation: crossover and mutation [Holland, 1975]. A canonical version of crossover is the *Single-Point Crossover*, described in the Background chapter of this thesis. My work uses variations on this method of crossover for floating point operations.

Wright discusses the implications of crossover on binary strings of data when the string is broken down into sub-sections each representing a *parameter* [Wright, 1991] . For instance, if the binary string represents four 8-bit integer numbers, there are four subsections corresponding to the four 8-bit integers. Should crossover choose a point in the middle of one of these sub-sections, it will introduce a *perturbation* into the crossover. This perturbation can be viewed as a mutation, as the resulting chromosome would likely not contain 8-bit integers that are in the union of the two parent chromosomes' 8-bit integer values. This is shown in Figure 3.4.

Figure 3.4: Perturbation Effect of Binary Crossover

base10-->				188								118							231							193					
Parent -->	1	0	1	1	1	1	0	0	0	1	1	1	0	1	1	0	1	1	1	0	0	1	1	1	1	1	0	0	0	0	1
base10-->				**188**								**121**							**24**							**62**					
Child -->	1	0	1	1	1	1	0	0	0	1	1	1	1	0	0	1	0	0	0	1	1	0	0	0	0	0	1	1	1	1	0
Parent -->	0	1	0	0	0	0	1	1	1	0	0	0	1	0	0	1	0	0	0	1	1	0	0	0	0	0	1	1	1	1	0
base10-->				67								137							24							62					

It is clear that the second sub-section of the child chromosome (the child chromosome is in the middle of the figure) contains the value 121. Neither of the parent chromosomes contain this value. This value was generated by the random nature of where the crossover point was placed (in this example, in the middle of the second sub-section). From this, we can conclude that the random nature

of the pure crossover function has the side effect of *perturbing* the data. This is an important feature of the canonical GA.

However, in a chromosome built from real-valued parameters, there are no sub-sections where this perturbation can be placed. In many applications where real-valued parameters need to be encoded using a binary encoding scheme, a real number would be encoded into a binary number, reducing its precision but allowing the encoding. Wright showed methods to deal with this loss of precision [Wright, 1991]. However, in order to accomplish this, we need to re-evaluate what the crossover and mutation operators actually do. Since this can be accomplished in a number of ways, I chose to use 5 different operations, each based in part on either the crossover or the mutation (or both) operations.

For this work, each chromosome is comprised of 24 real-valued parameters. This can be viewed as a point, or vector, in 24-dimensional space. For the following discussion, the examples show only 2 dimensions because its really hard to represent 24 dimensions with a picture!

Further, each parameter in the chromosome has a different *scale*. A set of six normalization parameters were chosen as representative of the base unit of scale for the six values in the PF vector shown in Equation (3.1). This normalization vector is used to express the scale of each type of potential field parameter. The values used are shown in Figure 3.5. This normalization vector is used by some of the generators when creating new random values for each of the PF vector values. These normalization values, multiplied by a configuration constant, are typically used as the standard deviation for Gaussian distributions around existing PF vector values.

Figure 3.5: Normalization Vector for Potential Field Parameters

Attractor		Repulsor		Spring	
Coefficient	Exponent	Coefficient	Exponent	Coefficient	Exponent
250	0.075	10,000	0.1	0.2	0.05

These chromosome generation operations, in the context of this algorithm, act to either explore the solution space or focus the search on one part of the solution space. Further, since the quality of the population of chromosomes cannot be hurt by exploring in a poor direction[4], the algorithm randomly selects which of the following generation algorithms is used each time a new chromosome is generated. The rest of this section describes the new chromosome generation operations.

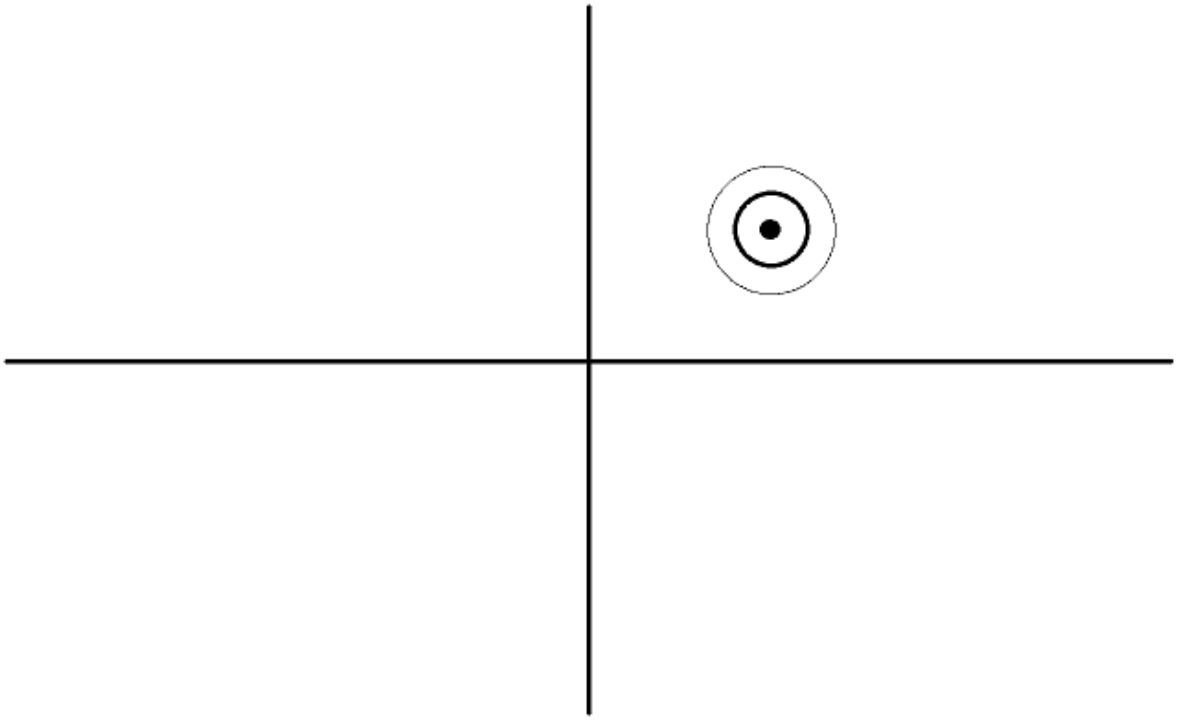
[4] *Poor Direction* referring to a direction that leads to lower fitness values

3.5.1 Uniform Mutation

In *Uniform Mutation*, a single parent chromosome is selected. Then, each component of the parent chromosome is mutated by some amount. The amount mutated is a Gaussian distribution centered on the parent value. The standard deviation is based on some constant times the normalization vector shown in Figure 3.5. For this work, the constant used was 1. This produces a new *PFGrid* in a manner very similar to the method employed by Larose, called "Random Shock" [Larose, 2006].

The resulting chromosome in 2d-space could be viewed as falling within the circles found in Figure 3.6. This type of mutation doesn't resemble the mutation operator described in Holland's work, as it acts on all members of the chromosome to move the point in N-dimensional space [Holland, 1975]. This type of chromosome generation was meant to focus on one particular area of the solution space. This is a form of simple *hill-climbing*.

Figure 3.6: Uniform Mutation in Two Dimensions



3.5.2 N-Point Mutation

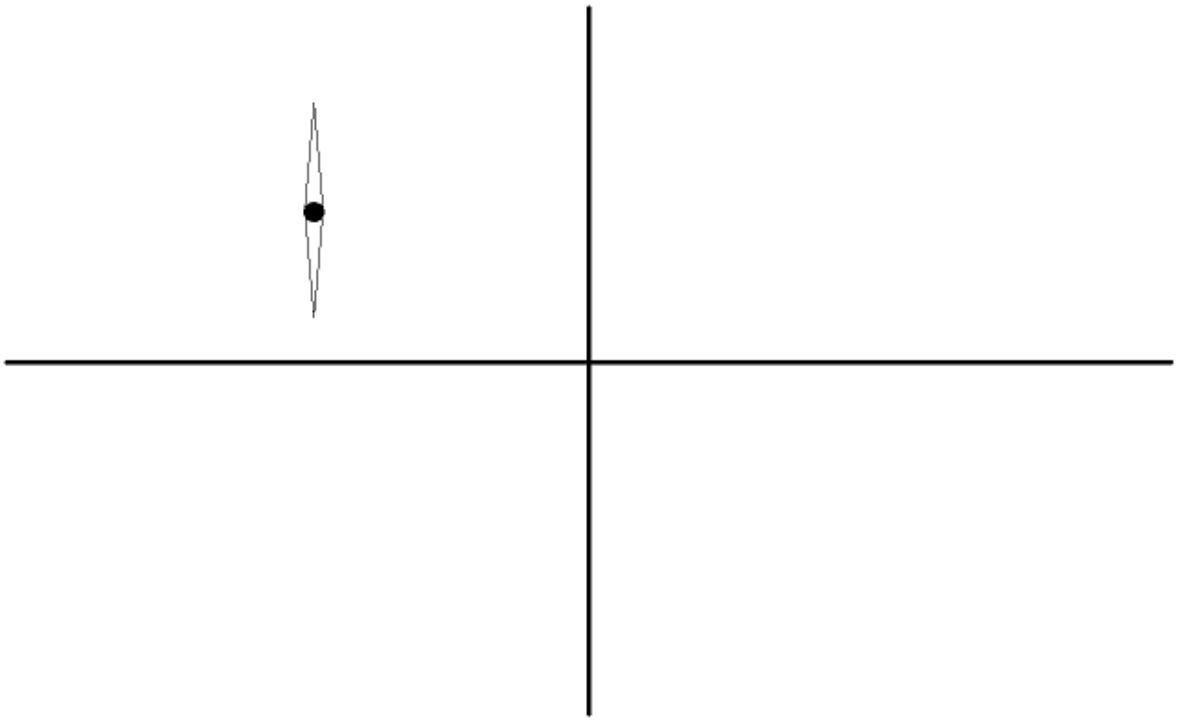
In *N-Point Mutation*, only a few of the parameters in the parent chromosome are mutated. The actual number of mutations N is based on a Gaussian distribution around a constant number with a small

standard deviation. For this work, 2.4 was the mean number for N (10% of the chromosome values), and the standard deviation was 0.6. Typically, 2 or 3 of the parameters in the chromosome will be mutated.

For each parameter that gets mutated, the same type of mutation as seen in *Uniform Mutation* is used. Namely, a Gaussian distribution centered on the parent value with a standard deviation based on a constant times the normalization vector component. For this work, the constant value that the normalization component was multiplied by was 10, yielding quite a large range of possible new values for the parameters selected.

The resulting chromosome after *N-Point Mutation* can be visualized in two dimensions using Figure 3.7. In this figure, one of the two dimensions was chosen for mutation, with the possible new values shown.

Figure 3.7: N-Point Mutation in Two Dimensions



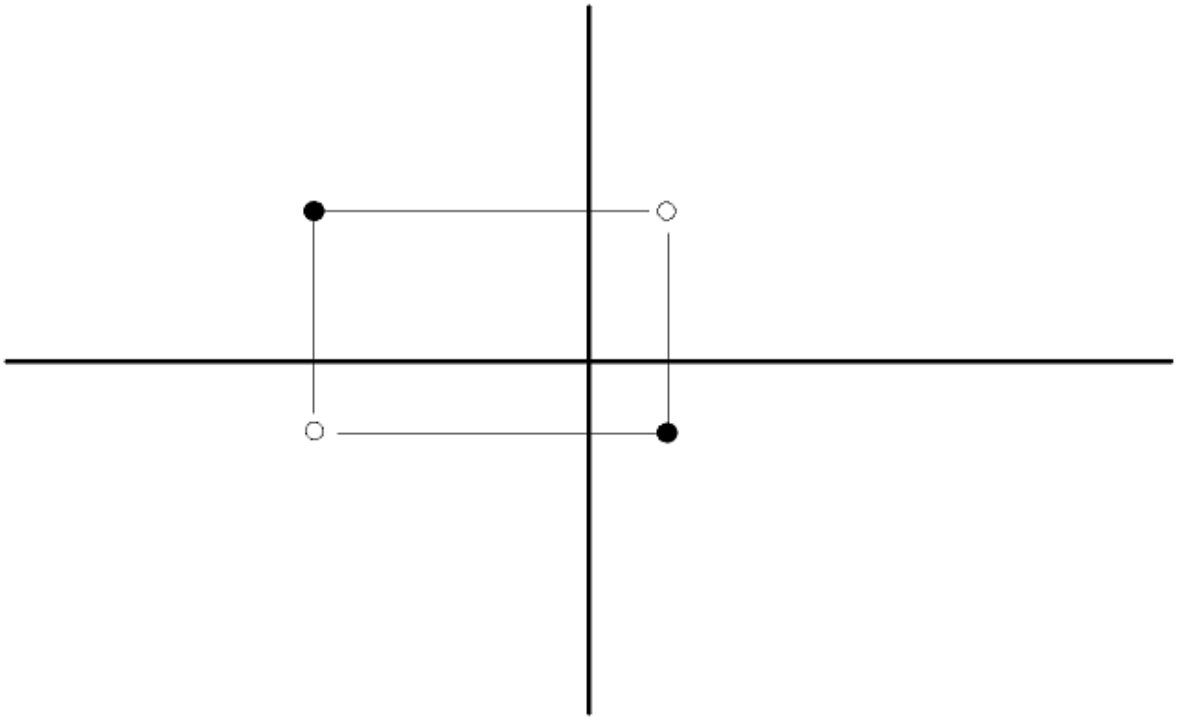
The mechanism behind this type of mutation most resembles the mutation operator described in Holland's work. It acts on only a specific small subset of the parameters of the chromosome to move it in the solution space. It is a hybrid of a focused and an exploratory operation. Empirical evidence

shows that it is particularly good at exploration (see Discussion). While this is a form of *hill climbing*, it acts to explore more than *Uniform Mutation* does. This is due to the high value used for the Gaussian distribution's standard deviation.

3.5.3 Uniform Crossover

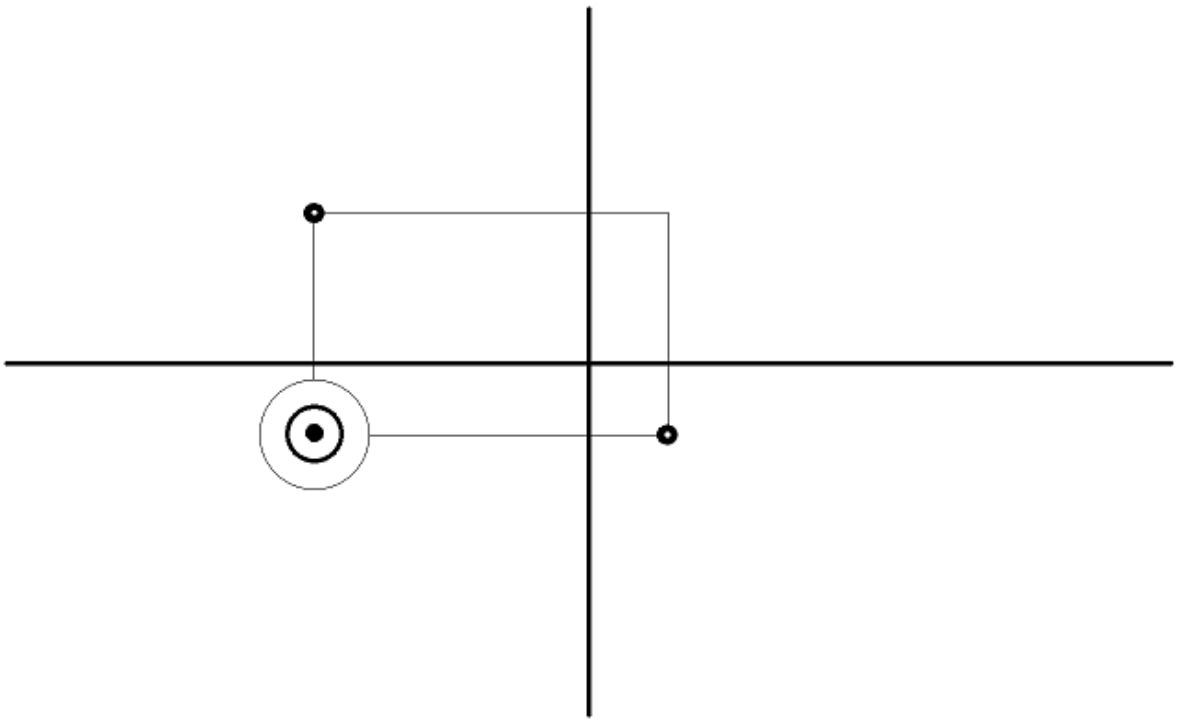
In *Uniform Crossover*, two parent chromosomes are combined to produce a new chromosome. Much like in uniform crossover for a canonical GA, each parameter of the resulting chromosome is taken from one or the other parent chromosome. The parent chosen for each parameter is a random operation.

Figure 3.8: Uniform Crossover in Two Dimensions



The basic operation of *Uniform Crossover* will result in a new chromosome that is on one of the corners of the hypercube defined by the parent chromosomes. This is shown for two dimensions in Figure 3.8. Since each of the parameters for the new chromosome must contain the parameters of one of the parent chromosomes, none of the natural perturbations shown in Figure 3.4 will occur. To compensate for this, at the end of the process of taking values from one or the other parent, the resulting chromosome is moved in the solution space using the same algorithm for *Uniform Mutation*, but with a smaller randomness (shown in Figure 3.9).

Figure 3.9: Uniform Crossover in Two Dimensions, with perturbation



Wright studied this type of operation in his research [Wright, 1991]. It is hypothesized that if the two parent chromosomes are both spatially close to the same *local maxima* in the solution space, that local maxima will have a high probability of lying within the circled region region of Figure 3.10. However, when the components of the parent chromosomes are combined into the new child chromosome using *Uniform Crossover*, the resulting chromosome will likely be outside the suggested circled region (shown by the hollow circles in Figure 3.10). This suggests that *Uniform Crossover* acts as an exploration operator, not a focusing operator.

3.5.4 N-Point Crossover

N-Point Crossover is a similar operation to *Uniform Crossover*, but with one significant difference. Its focus is to leave long chains of parameters from the parent chromosomes unaltered in the new chromosome. This crossover works in a similar fashion to its counterpart for canonical GAs. It can be seen pictorially in Figure 3.11.

The number of *crossover points* is a random function. As the algorithm scans the parameters of the chromosomes, a random number is generated in the range $[0, 1)$. If this random number is less

Figure 3.10: A Potential Effect of Crossover

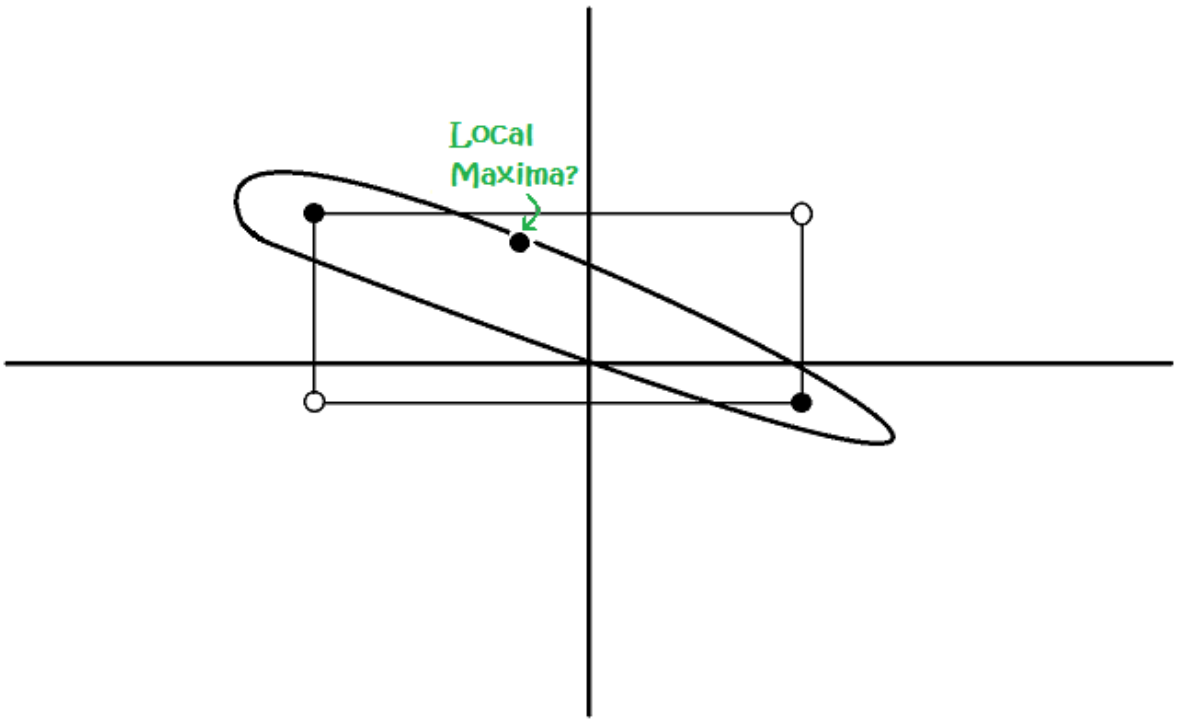
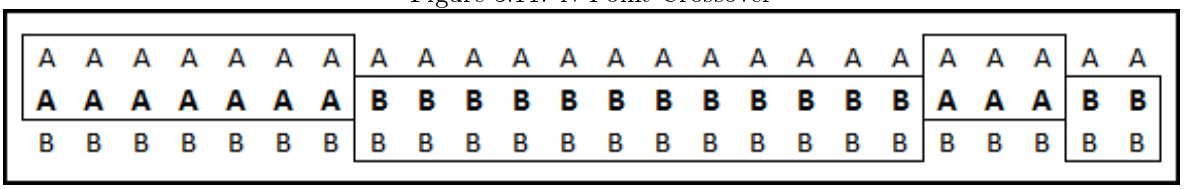


Figure 3.11: N-Point Crossover



than some percentage P, the algorithm switches which parent it takes chromosomes from when filling in the new chromosome. I used 10% for this parameter P. This may be clearer by looking at the pseudo code in Listing 3.5.

Listing 3.5: Pseudocode for N-Point Crossover

```
LET newChromosome = Blank_Chromosome
LET parentChromosome[2] = Parent_Chromosomes

LET source = 0
FOR i = 1 TO ChromosomeLength
    IF Random_Number(0,1) < Probability_To_Switch
        source = 1 - source
    newChromosome[i] = parentChromosome[source][i]
END_FOR
```

Both *Uniform* and *N-Point Crossover* act as exploration operations. *N-Point Crossover* was meant

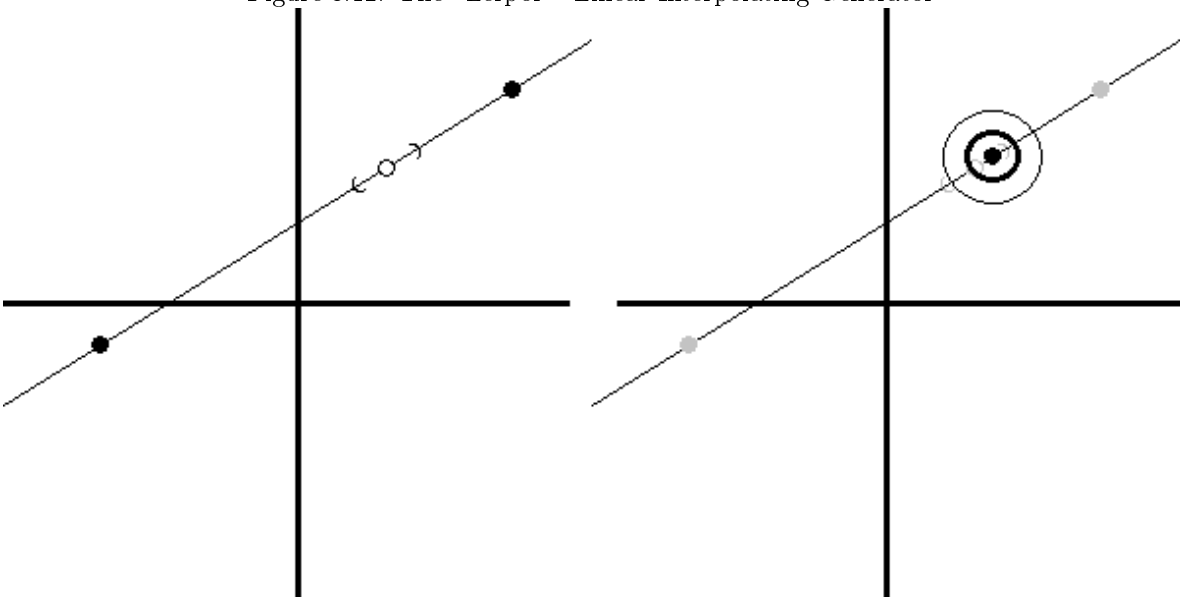
to act with less exploration and more focus. In practice, this was found to be generally true, but not to the degree that I has hoped.

3.5.5 Linear Interpolation

In the *Linear Interpolation* generator, which I call the *Lerpor*, I once again consider the idea presented in Wright's work and in Larose' work that the best solution may lie in the region between two points in the solution space (shown in Figure 3.10) [Wright, 1991, Larose, 2006]. Larose called this "Whole Arithmetic Crossover". The *Lerpor* does not have any obvious counterpart in the canonical GA.

Referring to Figure 3.12, we look for a point on the line joining the parent chromosomes. A point is selected on this line, biased by the fitness of each of the parent chromosomes. This point is closer to the parent with the higher fitness. Then, this point is moved in one direction or the other by some small amount. The amount moved is again a Gaussian distribution centered on the selected point with a constant standard deviation, in this work set at 0.1. This is shown in Figure 3.12 as the segment on the line between the parenthesis. Once this point is selected, a small *Uniform Mutation* is introduced, providing the new value for the chromosome.

Figure 3.12: The "Lerpor"- Linear Interpolating Generator



The *Lerpor* is designed specifically to be a focusing generator. It concentrates on pulling a good chromosome in the direction of another good chromosome, with the hope that a better solution lies

between them. Empirically, it has shown itself to provide good chromosomes more often when the population is still advancing at a rapid rate.

In this section, I described how chromosomes were generated. This section concludes the evolution part of my research. Next, I describe the co-evolution phase.

3.6 Co-Evolution

I found that my algorithm was effective in evolving a population that had high-performance chromosomes relative to the opponents that were used in the evolutionary process. However, the chromosomes evolved this way would generally do significantly worse against opponents that were not involved with their evolution. To account for this, I implemented a co-evolutionary algorithm, where two populations of chromosomes were evolved together and used each other to provide new opponents. This co-evolution algorithm is explained in this section.

Two separate populations were initialized with random chromosomes. These chromosomes, after initialization, were used to automatically generate epoch 1 for co-evolution. Epoch 1 denotes the first epoch of co-evolution, while epoch 0 is the first epoch for standard evolution.

At the beginning of each epoch, the co-evolution algorithm would select a set of chromosomes from each population to be used as opponents for the other population. Selection was based on the fitness of the opponents in the previous epoch. For this work, 5 opponents[5] were selected for each population.

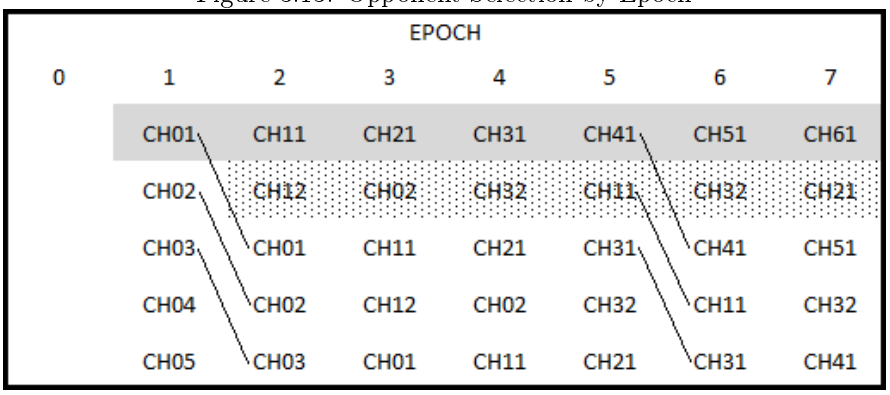
Except in epoch 1, when all 5 opponents came from epoch 0, the method of selection was straightforward. The first opponent was the best chromosome evolved in the last epoch. The third through fifth opponents were the first through third opponents from the previous epoch. The second opponent was randomly selected from all previous epochs such that it was not a duplicate of the ones already selected. This is depicted in Figure 3.13.

Using this approach, I was able to train each population against an ever changing set of opponents. This set of opponents had three notable traits:

Consistency: The majority (3 of 5) of the opponents were opponents the population had trained against before. This provided a sort of momentum in learning. When this consistency was not

[5]The number 5 was an arbitrary choice that balanced run times with maintaining a good representative set of opponents

Figure 3.13: Opponent Selection by Epoch



present in the opponent set, I found that the populations did not converge to good results as often, and it took them much longer to converge to good results.

Randomness: The introduction of the random element at index 1 (the row highlighted with dots) provides a couple of features. First, it allowed good chromosomes from early epochs to re-enter the training mix. This is similar to the *Hall of Fame* approach [Rosin and Belew, 1997]. Next, it allowed for an obvious degree of randomness in the choices for opponents. This helped prevent the co-evolution from over-specializing in only those most recent opponents.

Challenge: The new epoch had to train against the best chromosome from the previous epoch. This chromosome is assumed to be the best so far, although that may not be the case (consider the *rock-paper-scissors*[6] analogy). By selecting a chromosome in this way, I guaranteed that each epoch had at least one of its chromosomes present in the opponent set for at least three future epochs- more if the random element happened to be selected that epoch again in the future.

In addition to these 5 opponents, I also evaluated each chromosome against a *static opponent*. This static opponent did not move any of its vehicles. The vehicles simply shot back at any other vehicle that entered their firing range.

Since there were no expected terminal fitness levels for co-evolution, the algorithm was terminated at an arbitrary point in time. The algorithm advanced each population to the next epoch using a slight modification to the algorithm shown in Listing 3.3. In evolution alone, the *target fitness* was

[6] In the game of Rock-Paper-Scissors, there is no one best choice- the value of the choice is wholly dependent on the other player's choice.

determined by the initial chromosomes in only one population. However, in co-evolution, I used the higher *target fitness* from both populations, requiring both populations to reach a similar fitness level at each epoch.

The fitness level achieved during each epoch was relative to the populations being evolved. Since the populations were switching to new opponents at each epoch, the fitness of each chromosome was also relative to the epoch in which the chromosome was evaluated. To provide a more objective evaluation of how the algorithm performed over time, I wanted to see how the evolved tactics performed against opponents that had never been seen during evolution. To show this, I selected the best chromosome in each epoch from each population. For each of these chromosomes, I evaluated its performance against a set of 3 opponents that were never encountered during evolution, using a scenario (vehicle layout) that was also never seen during evolution. With this methodology, I felt like I could empirically show that the algorithm did produce generalized tactics and that these tactics performed better over time relative to opponents that had never been seen before.

3.7 Summary

In this chapter, I explained the methodology used to evolve the parameters to potential fields such that these potential fields can be used effectively as implicit tactics in an RTS game. In the next chapter, I discuss the results of my experiments. I also comment on why certain phenomena were present in the evolutionary process.

Chapter 4

Results and Discussion

There were two phases to this research. First, create an evolutionary environment where a tactic can be learned against an opponent. Second, create a co-evolutionary environment where a tactic can learn to generalize against other opponents. In this chapter, I discuss the RTS game scenarios that were used in evolution and co-evolution, followed by the results of both evolution and co-evolution.

4.1 RTS Game Scenarios

For all of the experimentation I selected two game scenarios, each with six identical vehicles per player. The layouts of these scenarios are in Figures 4.1 and 4.2. This choice was somewhat arbitrary. I did want to have a variety of initial *grouping configurations*, but this didn't seem to affect the results. The evolved tactics take care of grouping implicitly after each game starts.

For each scenario, Herkermer played 6 games using chromosomes from the population being evolved. These games used different combinations of scenario layouts and sides (e.g. Scenario 1 / Player 2, etc). I also created two types of opponents:

Static Agent: The *Static Agent* simply sets the *desired speed* and the *desired heading* to zero. This agent merely provides the opponent with a set of targets that shoot back. It was used to see how well evolution performs against static targets. These types of targets could be viewed as gun turrets- stationary guns that merely fire on enemies as they approach. This served to help guide the initial creation of $PFGrid$ values for populations. If a new, randomly generated $PFGrid$ could not *complete* a game against this type of agent, it was not included in the population. This would be the case if a new $PFGrid$ caused the vehicles to simply run away from enemy vehicles. Running away is clearly not conducive to the overall goal of *eliminate your opponent.*

Figure 4.1: First Scenario for Experiments

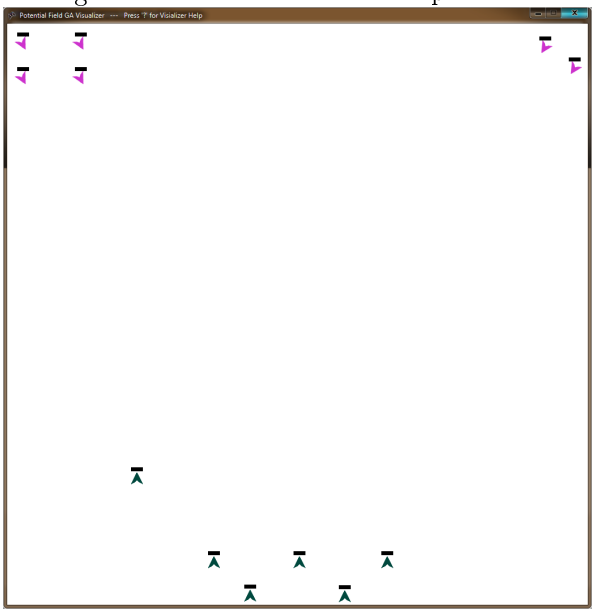


Figure 4.2: Second Scenario for Experiments

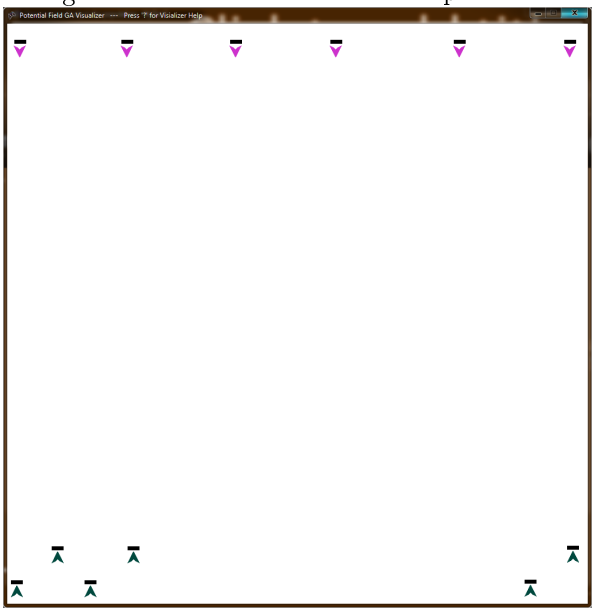
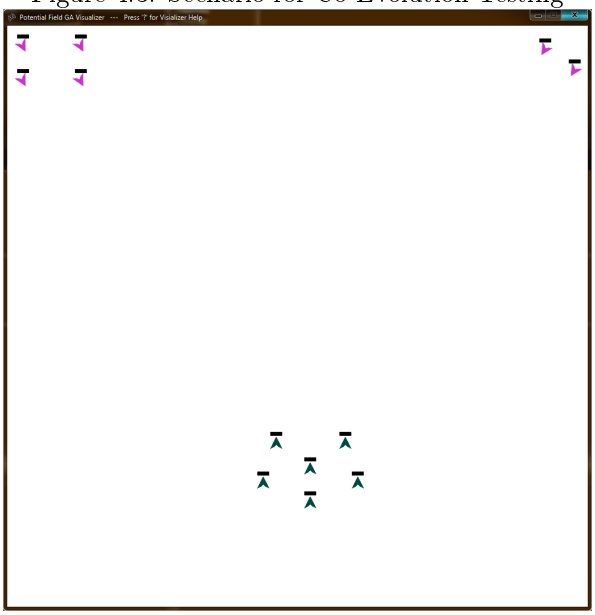


Figure 4.3: Scenario for Co-Evolution Testing



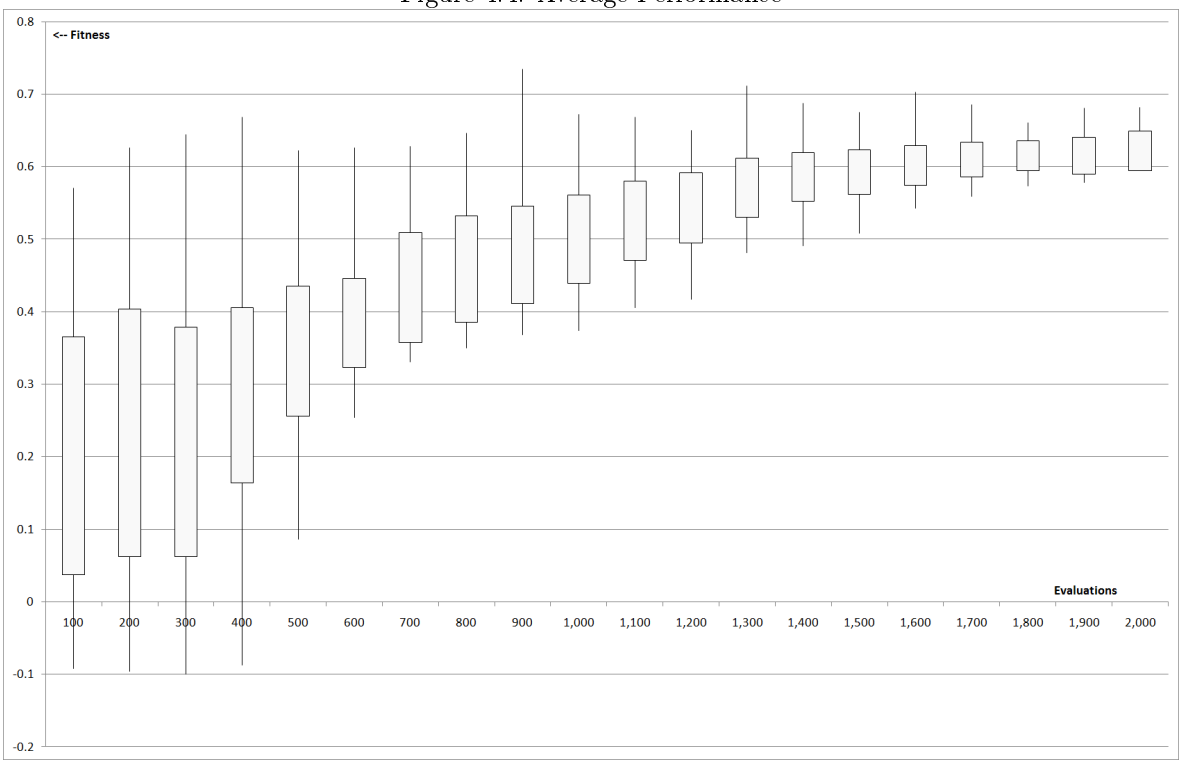
Hand Crafted: I created a set of hand-crafted values for a *PFGrid* that was used to configure a Herkermer agent. I hand-crafted these values without any view towards optimization. With these values, I wanted a high degree of confidence that vehicles wouldn't collide with other vehicles that are on the same team. I also wanted them to be effective enough to simply win a game against a *Static Agent*.

For co-evolution, I created a third scenario shown in Figure 4.3. This provided different starting positions for the players. It allowed me to show how well a tactic was adapted to a scenario that was never seen during the co-evolution process.

4.2 Evolution

The process of evolution to improve tactics against a specific set of opponents went well. Figure 4.4 shows the progress made by the evolutionary algorithm over the course of time for 10 runs of the evolutionary algorithm. The picture shows that the algorithm followed what should be a recognizable pattern for learning algorithms of this type. The graph shows the fitness minimum, maximum, and average plus and minus one standard deviation through time. Each point on the X-axis represents the information for the chromosomes that were generated during the interval indicated by the X-axis. This explains why the maximum fitness is higher than future fitnesses in the interval from 300-400, the

Figure 4.4: Average Performance



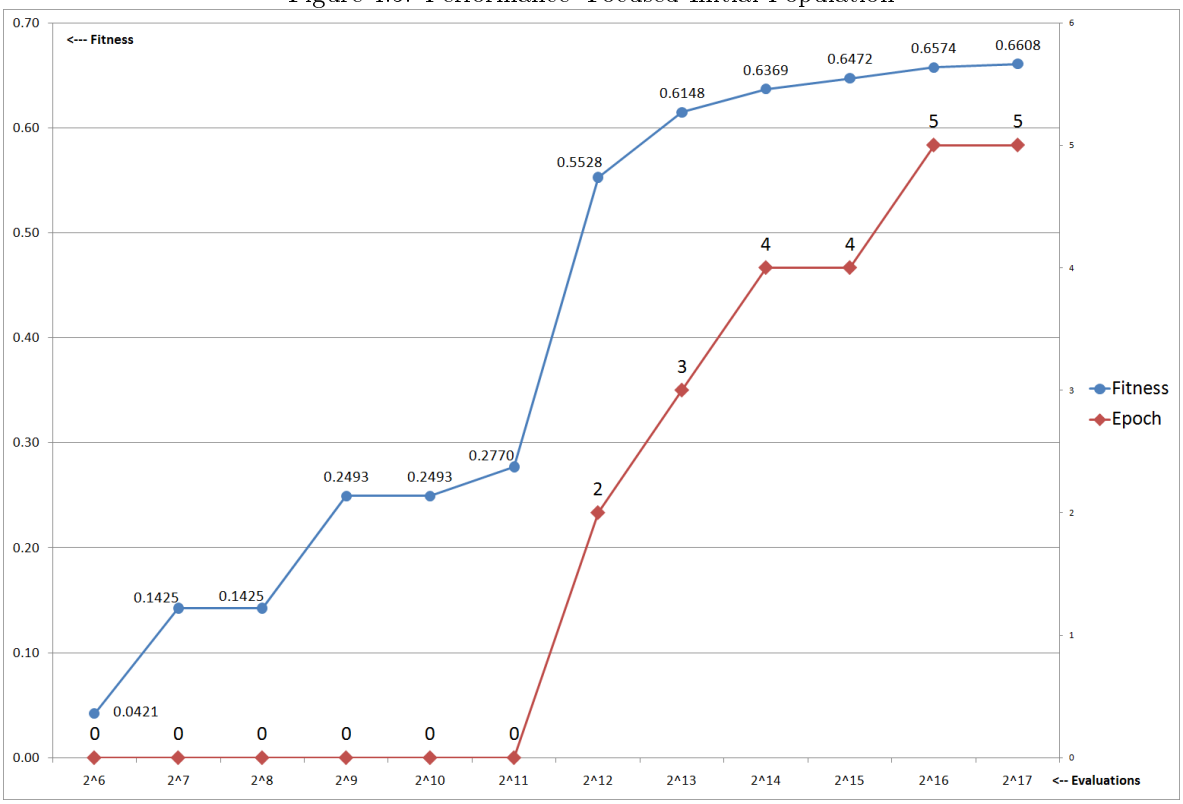
interval from 800-900, etc. The averages (the boxes on each vertical line) contain the more meaningful data.

In Figures 4.5 and 4.6, I show the progress made by the evolutionary algorithm over the course of time for a single run of the evolutionary algorithm. In these graphs, I show the epoch that corresponded to each point on the fitness line. This gives some indication of how a typical run of evolution looks. The graphs show the progress based on initial populations which were generated in two different ways. The scenarios (number and layout of vehicles) for these populations were also considerably more complex than the scenarios used for the averages found in Figure 4.4.

The reason I chose two different initial population generation schemes was to explore how the algorithm worked under various starting conditions. In one of the populations, the initial values were more focused on an area of the search space that was known to contain a tactic that would simply win against a static opponent. In the other population, the initial values were selected with few conditions placed on the parameters.

In Figure 4.5, the initial 100 chromosomes of the population were generated based loosely on a

Figure 4.5: Performance- Focused Initial Population



set of hand-crafted potential field parameters. The hand-crafted parameters were designed such that they would win a scenario against a Static Agent, but with a low score. Each of the 100 initial chromosomes for this population was generated by repeatedly mutating this hand-crafted chromosome using the *Uniform Mutation* algorithm found in Section 3.5.1. The degree of mutation (the standard deviation for the Gaussian distribution) was ten times the normalization values found in Figure 3.5.

For the second population, shown in Figure 4.6, the initial 100 chromosomes were generated using a uniformly random distribution of values for each parameter in the chromosome. The limits placed on each parameter are shown in Figure 4.7. In all of the testing performed, values near the edges of any of these ranges were never found in an acceptable chromosome.

When Figures 4.5 and 4.6 are compared, they show graphically that the evolutionary algorithm performed approximately the same regardless of the type of initial population. The shape of the fitness line, plotted on a logarithmic scale, shows the learning progressed in a similar fashion to many machine learning algorithms. This was encouraging to me that my algorithm was an effective machine learning algorithm.

Figure 4.6: Performance- Uniformly Random Initial Population

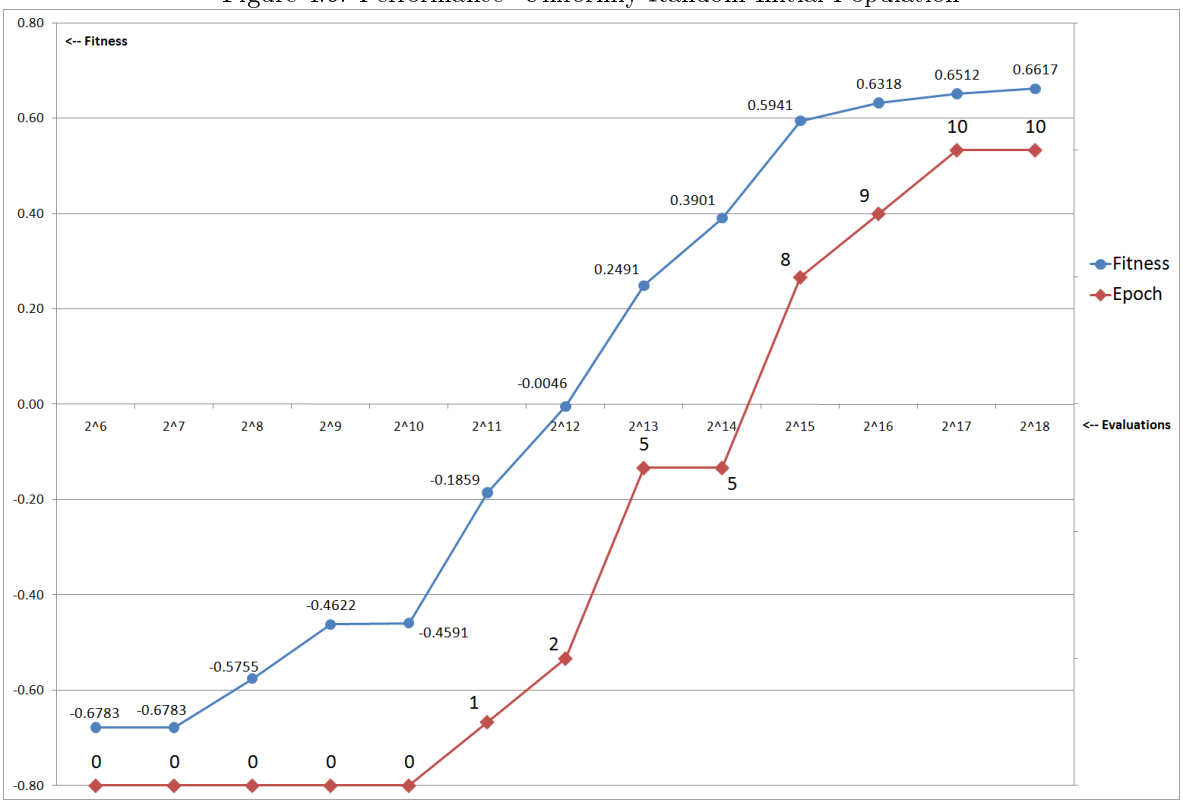


Figure 4.7: Limits placed on Parameters for Uniform Random Generation

Attractor		Repulsor		Spring	
Coefficient	Exponent	Coefficient	Exponent	Coefficient	Exponent
0 to 50,000	-6.5 to 0.01	-30,000 to 0	-8 to -0.1	0 to 30	0.0001 to 4

Some statistics about the two evolved populations shown in Figures 4.5 and 4.6 are shown in Figure 4.8. An interesting statistic in this table is the last one. Looking at the average distance between the best chromosome and the rest of the chromosomes with fitness above zero, we see distances in the 200-300 range. However, the best chromosomes found in the two respective populations had a distance metric of 3,147 between them. This represents the fact that these two evolutions produced two significantly different tactics, but each tactic performed roughly equally as well.

One can also see that the uniformly random initial population took roughly 225,000 evaluations to reach a maximum fitness of approximately 0.66. In the population with the focused initial chromosomes, approximately the same fitness of 0.66 was reached after around 124,000 evaluations. It took the evolutionary algorithm just under twice as long to reach the same fitness level for a much more randomized initial population.

Figure 4.8: Statistics for Evolved Populations

	Default Initial Population	Uniformly Random Initial Population
Average Total Chromosomes In Population	2,428	5,318
Average Total Chromosomes Evaluated	123,630	225,231
Average Epochs	5	10
Average Best Fitness	0.6608	0.6617
Average Fitness, Top 100 Chromosomes	0.6527	0.6523
Average Distance From Top Chromosome, Best 100 Chromosomes	262.49	39.53
Average Distance From Top Chromosome, All Fitnesses > 0	217.33	324.80
Average Distance between Two Best Chromosomes	3,147	

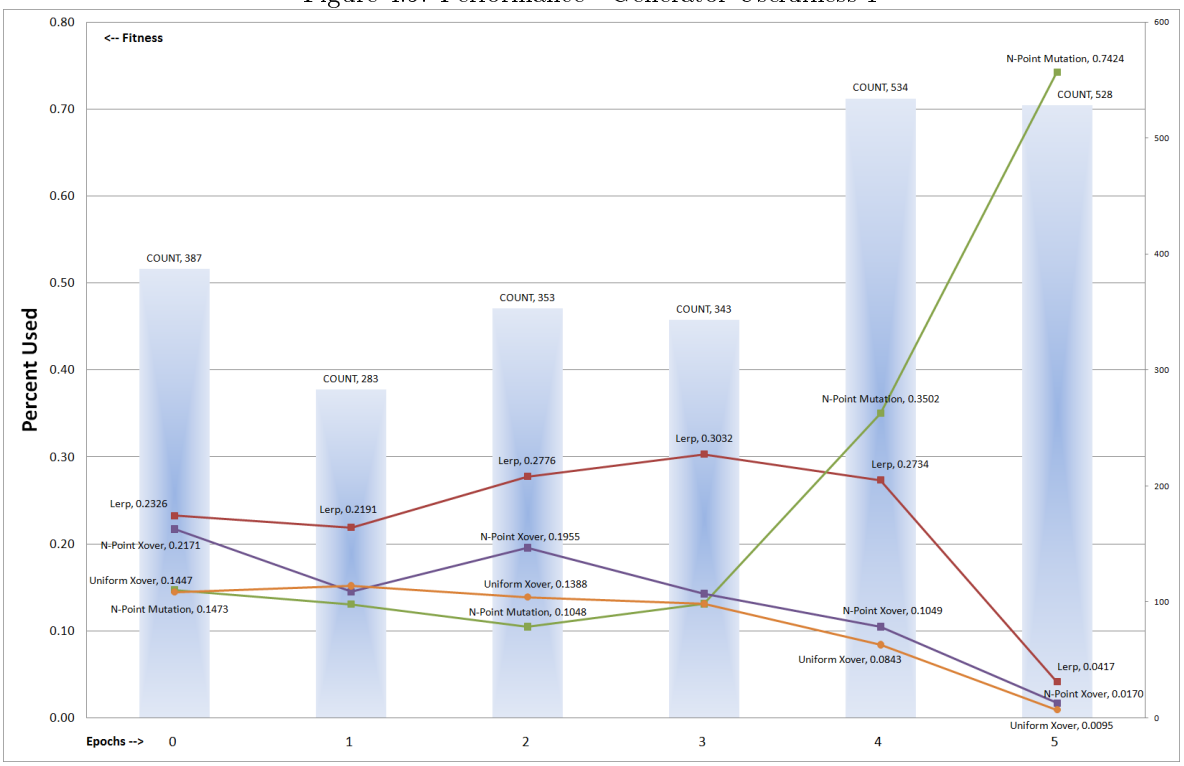
Another surprising statistic was the spatial diversity of the top 100 chromosomes in the uniformly random population. I expected this number to be much higher, as was seen in the diversity in this population for all fitnesses above zero. However, this number was relatively small. I hypothesize that this is because the population converged to one area of the solution space at the end. This may be because the majority of the initial chromosomes ended up in areas of the solution space where the local maxima were not very good.

4.2.1 Local Maxima

While these graphs are important to show that the evolution did indeed increase the fitness of the population over time, they do not show an interesting phenomenon that occurred in some of the evolution trials. Common to any search algorithm that is not exhaustive is the problem of *local maxima/minima* [Russel and Norvig, 2003]. Local maxima are areas of the solution space where changing the input vector by a small amount in any direction will not be able to find a higher fitness value, but a higher fitness value does indeed exist somewhere else in the solution space. In order to get around local maxima, an evolutionary algorithm must be able to explore an area of the search space that is relatively far away from where the search is currently concentrating.

Canonical *hill-climbing* algorithms are notorious for getting stuck at local maxima. There are many techniques available (simulated annealing, random restarts, etc) that try to overcome this problem. I believe that a hallmark of a good search algorithm is that it can adapt to its current situation (stuck or not stuck on local maxima) in order to explore as much of the solution space as possible.

Figure 4.9: Performance - Generator Usefulness 1

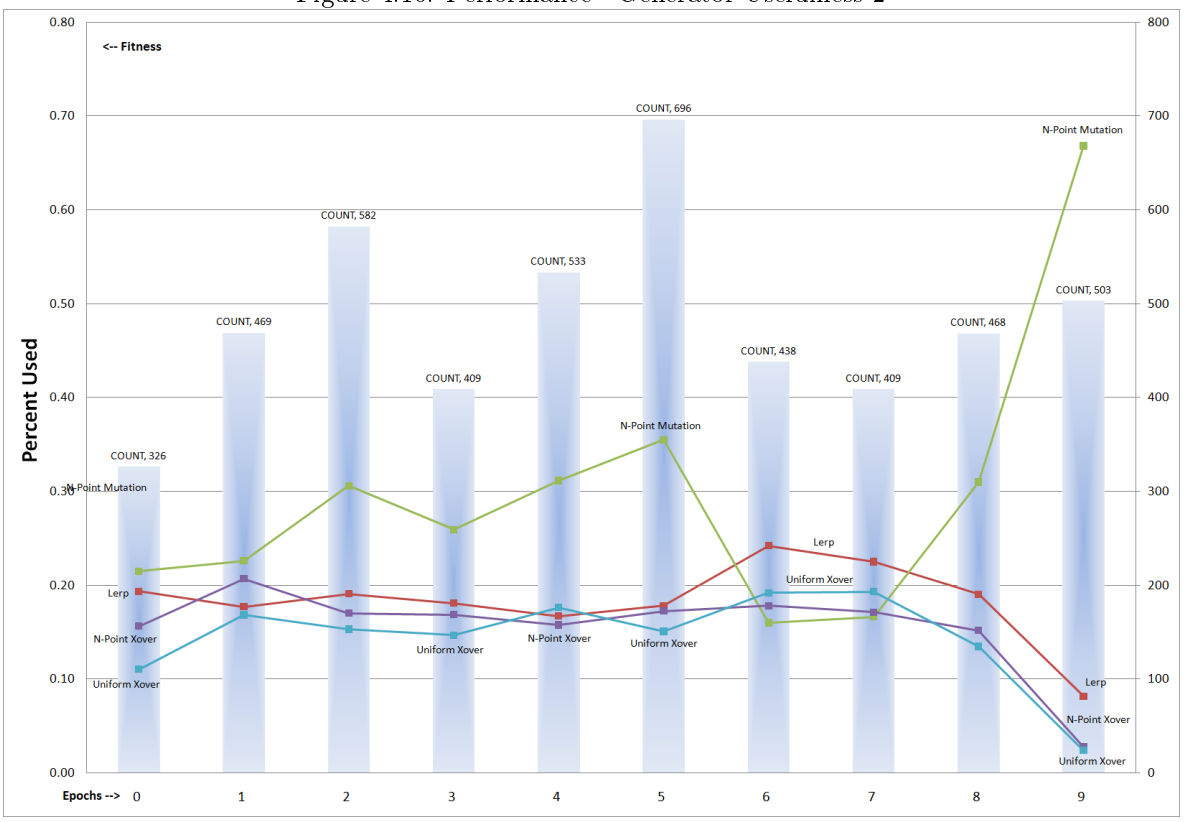


Consider Figures 4.9 and 4.10. These graphs show how often the use of each of the five generators resulted in a new chromosome being added to the population. Superimposed on the graphs is the chromosome count for each epoch. By looking at the super-imposition of the chromosome count, it is clear that the *N-Point Mutation* chromosome generation operation was used heavily in those epochs with the most chromosomes. In these epochs, relatively many chromosomes were generated in order to advance the population to its next fitness target. I hypothesize that it was difficult to find a better alternative to a local maxima in these epochs. *N-Point Mutation* was the generator that was most successful in advancing the population by the largest amount during these epochs.

These empirical results show that my algorithm was adaptive to the current state in the search and was able to overcome local maxima given enough time. It showed that, regardless of the composition of the initial population, the algorithm was able to evolve chromosomes that provide good tactics against those opponents that were used in the evolutionary process.

However, these tactics did not do as well against opponents that have never been seen before. Consider the population evolved whose fitness is plotted against epochs in Figure 4.11. The figure

Figure 4.10: Performance - Generator Usefulness 2



shows the range of fitness values for each epoch, with the range plus or minus one standard deviation above and below the average for each epoch highlighted. This population was evolved to a relatively high maximum fitness of 0.894. However, the following graph in Figure 4.12 shows that it did worse against the *hand-crafted* tactic. While the *best* chromosomes in each epoch were able to beat the hand-crafted opponent, the majority of them[1] could not beat it. Further, the best chromosomes could not beat the new opponent by as much as they beat the opponent they were trained against. I expected this as the outcome for evolution conducted in this manner.

4.3 Co-Evolution

The process of using co-evolution to generalize tactics went well. The tactics evolved were able to generalize and win games, but they were not able to win these games by the same margin as those evolved against specific opponents when played against those opponents. I expected this result, as a

[1] In Figure 4.12, there were chromosomes that represented tactics that could not complete a game against this new opponent. Those chromosomes were removed from the population.

Figure 4.11: Performance - Specific Opponent

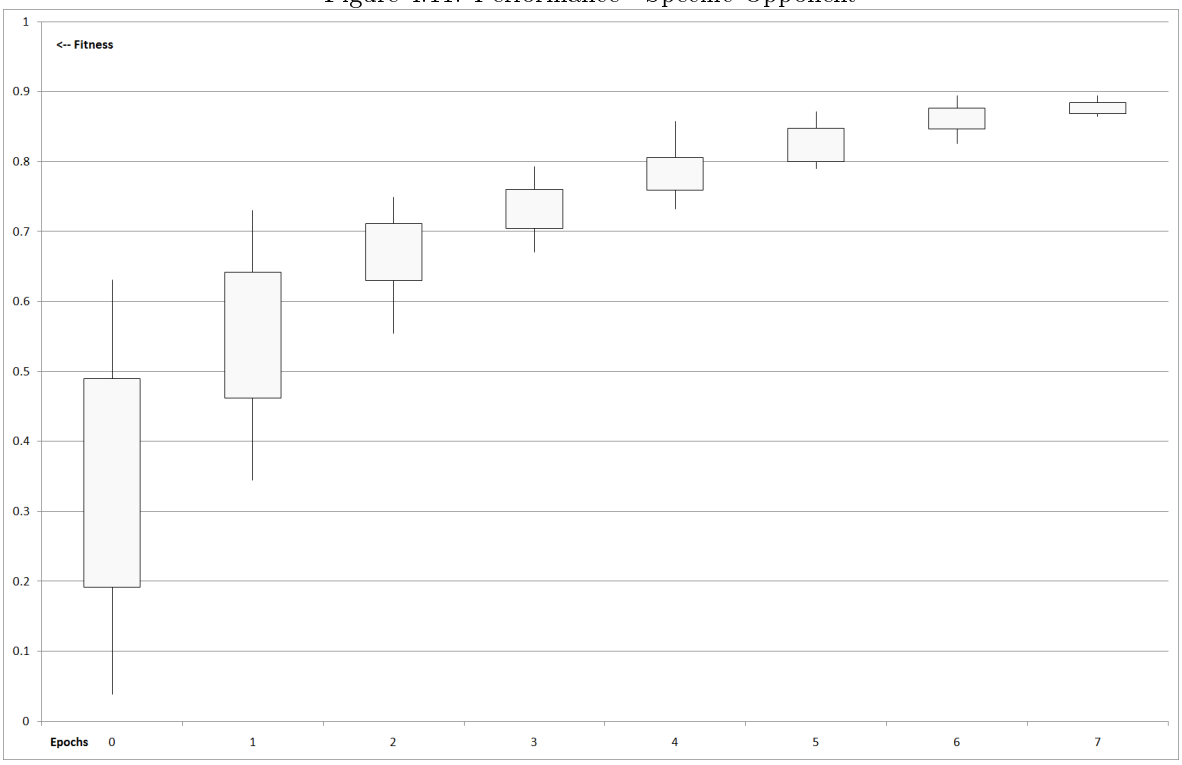
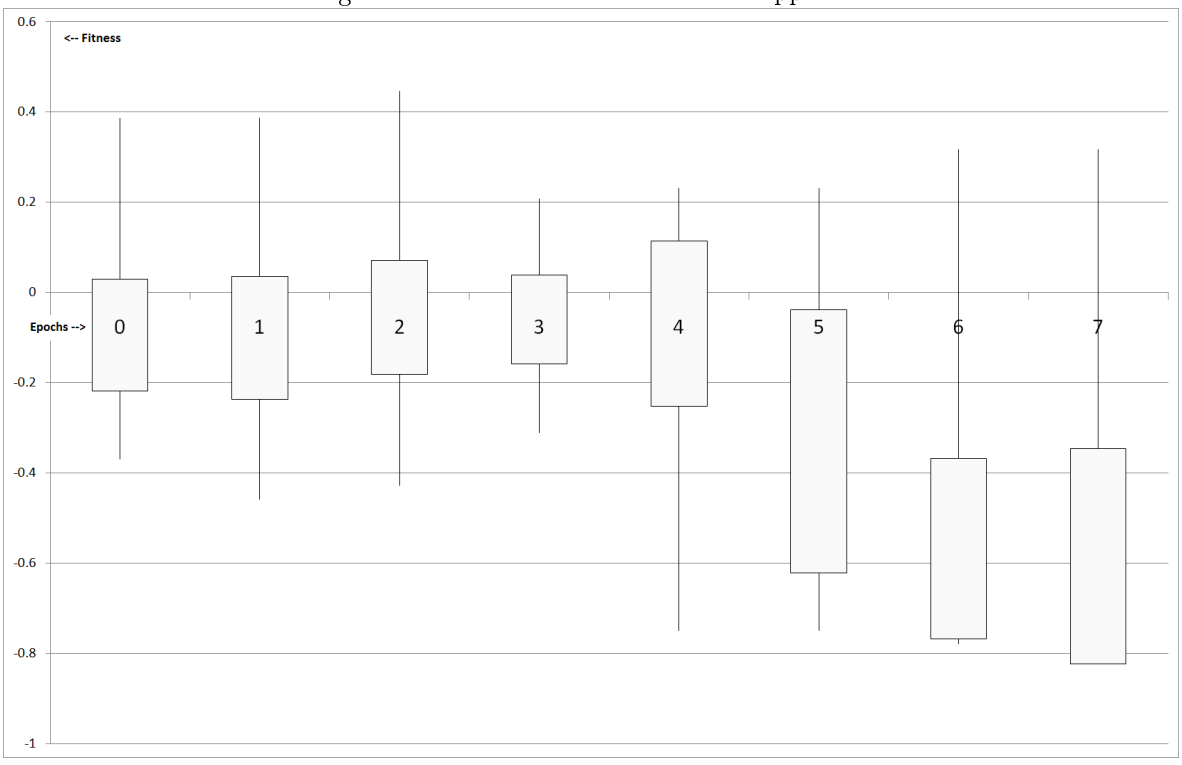


Figure 4.12: Performance vs. Unseen Opponent

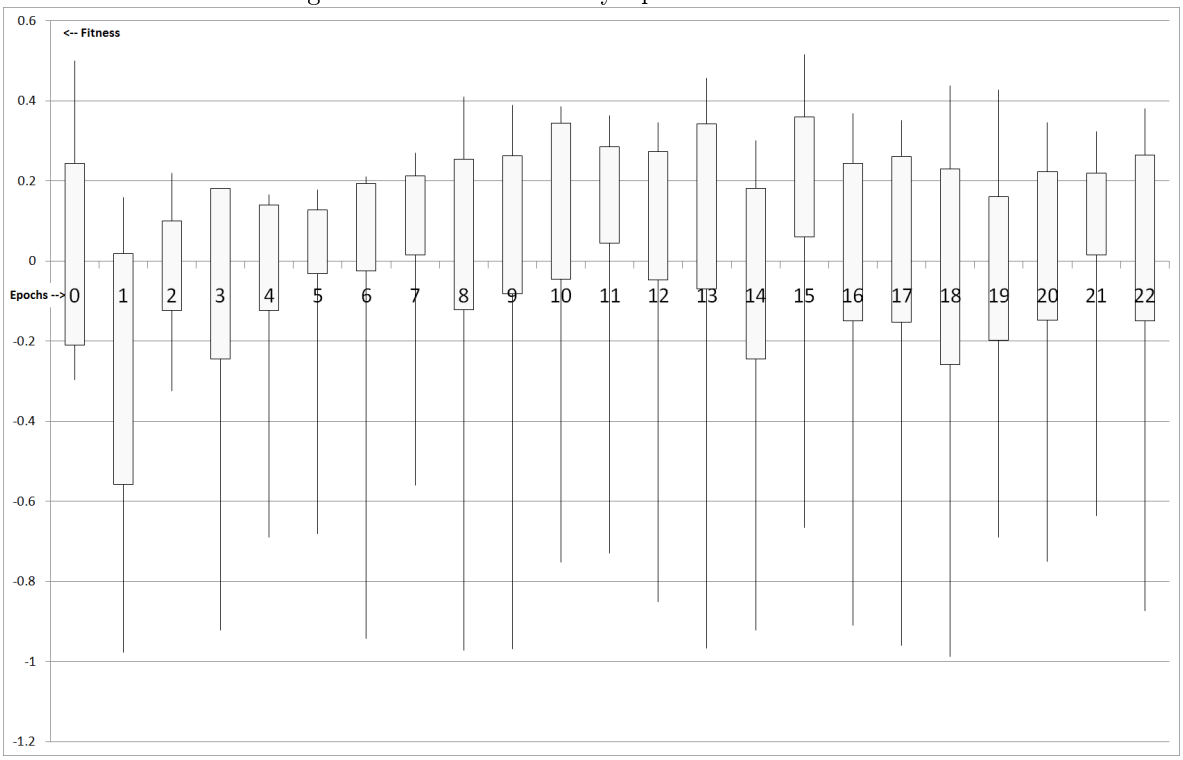


specialized agent would intuitively do better at what it specializes in.

For co-evolution, the fitnesses of the best chromosomes in the population through time were not very meaningful. Since each population was continuously being evaluated against a different set of opponents, the fitnesses through time did not show the increases found in evolution against a specific opponent. This is shown in Figures 4.13 and 4.14. In Figure 4.13, we can see that the fitness range for each epoch was large. We can also see that the maximum fitness did not grow with time. I didn't expect these values to grow, as they were always relative (epoch to epoch) to a new set of opponents. At each epoch, the algorithm had to re-evolve the chromosomes based on the new opponent set.

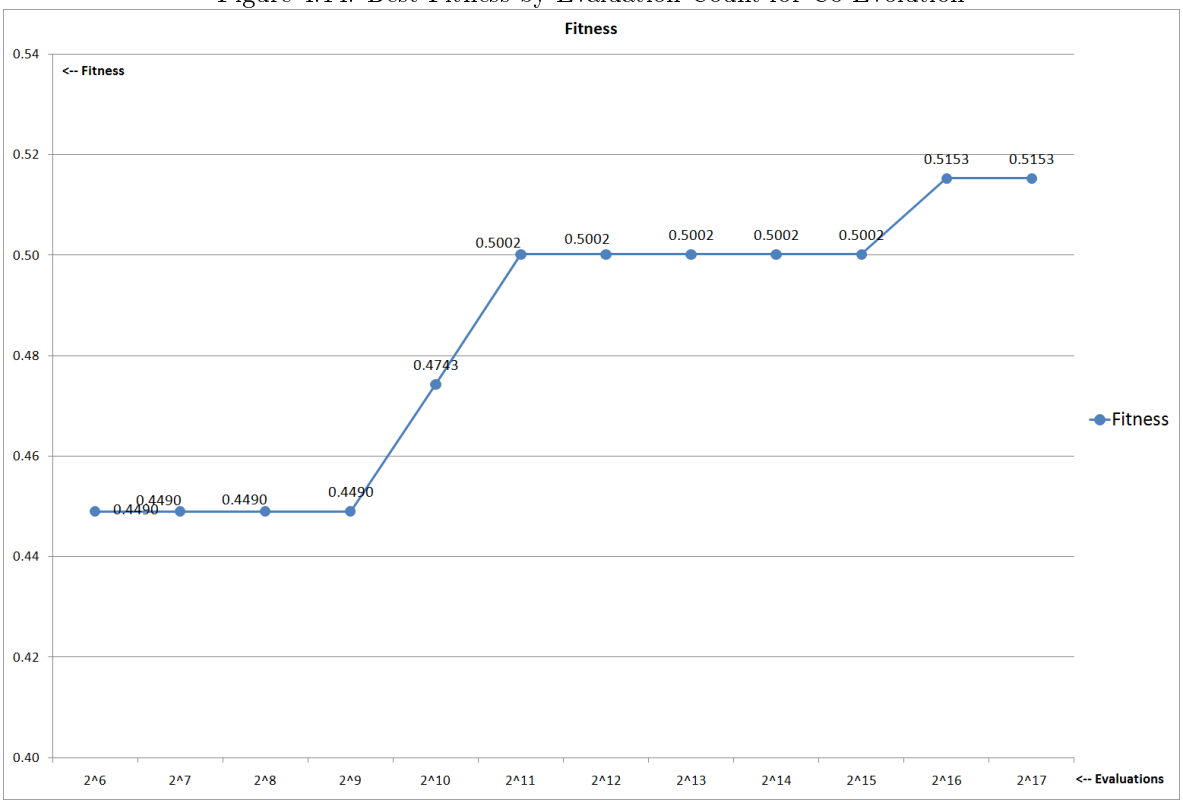
The data in Figure 4.14, showing the maximum fitness in the population over time, is not meaningful for a different reason. Over time, the populations' opponents will change. In some epochs, a population may be opposed by relatively weak chromosomes[2]. When evaluated against these weak opponents, one would expect high fitness values. If this happens early in the process of co-evolution, the maximum population fitness will reflect the fitness against an early weak opponent. This explains why Figure 4.14 looks the way it does.

Figure 4.13: Performance by Epoch for Co-Evolution



[2]Relative to the opponent chromosomes in other epochs

Figure 4.14: Best Fitness by Evaluation Count for Co-Evolution



For the purpose of evaluating the tactics resulting from co-evolution, I selected a *baseline* against which the best chromosomes from each epoch could *test* themselves. This was accomplished by creating an environment that was not seen during co-evolution and evaluating the chromosomes from each epoch of co-evolution using this new environment. I chose a third scenario (vehicle layout), shown in Figure 4.3, that was not used during co-evolution. I then chose three opponents that had not been used during co-evolution[3]. These opponents are described here:

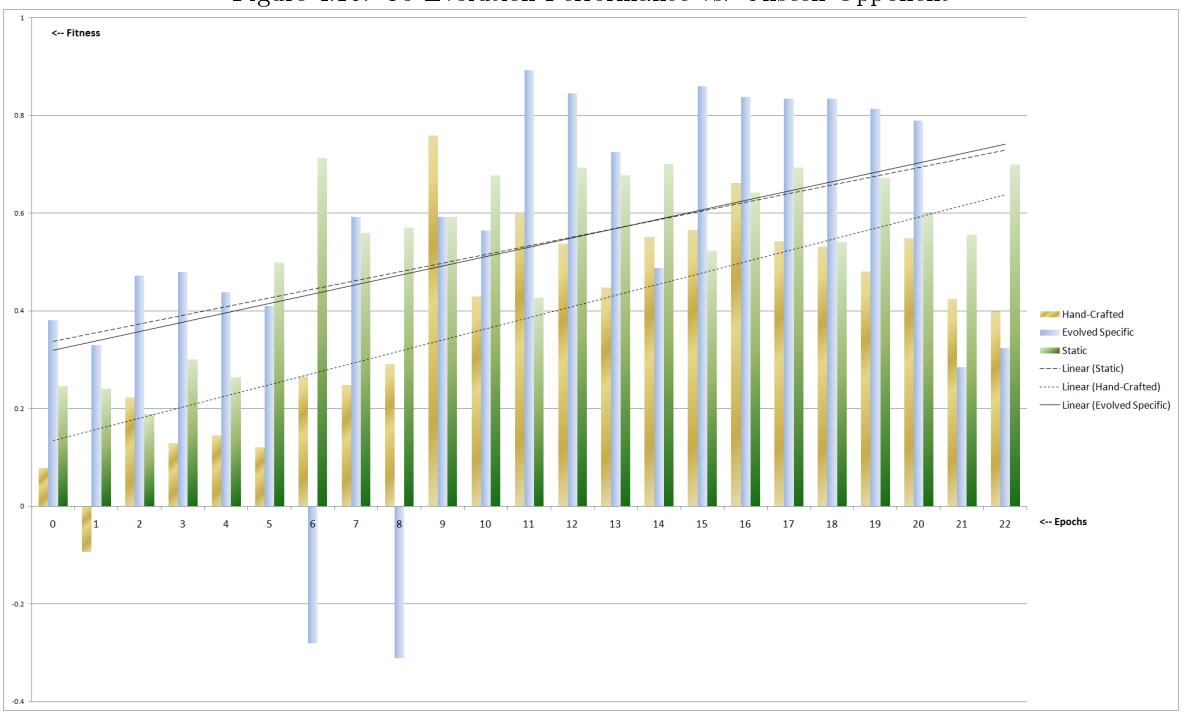
Static Opponent: This opponent did not move any of its vehicles. Its vehicles merely fired at any enemy vehicle that came into its firing range. A similar opponent was used during co-evolution, but with a different vehicle layout. I felt that changing the positions of stationary targets was sufficiently different to warrant including this opponent.

Hand-crafted Opponent: The hand-crafted tactic described in Section 4.1 was used as an opponent. This hand-crafted tactic was not used during co-evolution.

[3]It is statistically possible that two of these three opponents could have evolved naturally during co-evolution. These two mentioned here were checked against the chromosomes in the populations, and they were not found.

Specifically Evolved Opponent: The tactics used by this opponent were evolved against a specific and separate set of scenarios and opponents. These tactics were the best tactics evolved during an external evolutionary procedure.

Figure 4.15: Co-Evolution Performance vs. Unseen Opponent



The results of playing the tactics described by the best chromosomes from each epoch against the above described opponents are shown in Figure 4.15. Epochs are used in this graph because the opponents changed for the populations when new epochs were generated. I am showing the progress made by the co-evolution process at every point when it switched opponents for the populations. This data is significant because it shows that the co-evolution did increase the performance of the tactics against previously unseen opponents and situations.

4.4 Video of Results

I have selected a set of tactics to publish in video form. These have been posted to YouTube. These videos are shown in Figure 4.16. In that Figure, *Player 0* starts at the top of the screen and *Player 1* starts at the bottom. The *HC* opponent was the hand-crafted *PFGrid* used with Herkermer, described in Section 4.1.

4.5 Summary

This chapter contains the results of my research. I showed the RTS game scenarios that I used for evolution and co-evolution. Then I showed how the process of evolution improved the fitness of the chromosomes in the population over time. Finally, I demonstrated how using co-evolution provided stronger tactics over time when played against opponents never seen during co-evolution, even though the overall fitness of the chromosomes in the populations didn't grow at the same rate.

Figure 4.16: Videos with YouTube Links

YouTube Link	Player	Oppon't	Score	Notes
http://www.youtube.com /watch?v=GafZ7GLUBvU	0	Static	-1	Horrible Tactics. Merely allows vehicles to crash into each other. No significant formations
http://www.youtube.com /watch?v=wsNlZ6GWz5A	0	Static	-.4083	A little better at staying away from team-mates, but still collides with them. Very suicidal against enemies
http://www.youtube.com /watch?v=c-PEtvZbZog	1	HC	.0549	Started recognizing formations- couldn't quite get them down though. Doesn't want to collide with own vehicles, but will ram opponent
http://www.youtube.com /watch?v=hBPwh5z9eao	1	Static	-.1344	Same tactic as above, but against a static opponent. The same-team collision in the bottom left corner lost the game for this tactic. Tried to get formation down, but very sloppy. Still suicidal towards opponents.
http://www.youtube.com /watch?v=aVZ_y69LcUg	0	Static	.8479	A very successful tactic, but slow to execute. Did not create formations very quickly. It did do very well in protecting its hurt units.
http://www.youtube.com /watch?v=YzvAegkFw2Y	0	HC	.6124	A fast game against the hand-crafted opponent. So fast that formations weren't observable in this game.
http://www.youtube.com /watch?v=nUm9D8KHhPs	0	Static	.8462	The best chromosome from a uniformly random initial population. Formations were "loose". Focused on protecting weak units.
http://www.youtube.com /watch?v=1Rb4NREfRjY	0	Static	.8469	The best chromosome from a focused initial population. Stronger notion of formation in this one. The tighter formation allowed it to pick off the fourth vehicle in the bottom left before moving across. Lucky that the hurt unit didn't die at the end- A longer game may have provided different results.
The focused initial population provided a "prettier" formation, but it didn't seem to be as worried about its hurt units.				
http://www.youtube.com /watch?v=kZU3q5bDAik	0	HC	.5906	The best chromosome from a uniformly random initial population against the hand-crafted Herkermer tactic. It grouped closely and forced a same-team collision on the opponent's team.
http://www.youtube.com /watch?v=noGnvjV9qmA	0	HC	.5315	The best chromosome from the focused initial population. This one did not group as tightly around the enemy, allowing the enemy to get in more shots against it.
The following three were evolved against a specific scenario with a static opponent.				
http://www.youtube.com /watch?v=0IuToUbQmsY	0	Static	.8942	The scenario it evolved against / specialized in. This seemed very cautious. It formed tight formations. No collisions. It seemed aware of hurt vehicles.
http://www.youtube.com /watch?v=Br5UjsGAddo	0	Static	.6189	Against a different initial layout of vehicles. It lost a significant amount of fitness due to changing the vehicle layout. It seemed very interested in formations. It did have a same-team collision in tight quarters. It did protect weaker units.
http://www.youtube.com /watch?v=beCwLKLV6dQ	0	HC	.4466	Against the hand-crafted Herkermer agent. It seemed very aggressive. It did try to surround the opponent, but it lost half of its vehicles in the process.
The following were the results from co-evolution				
http://www.youtube.com /watch?v=qndzj5WCE2A	1	Spec	.8347	Used a bit of deception to draw fire while others swarmed in.
http://www.youtube.com /watch?v=f3-sG_JBN_c	1	Static	.5765	Didn't use the formation very strongly. It did seem like hurt vehicles ran away from even their friends.
http://www.youtube.com /watch?v=hVHctL_2WYY	1	HC	.5424	In this case, running away from friends allowed the hurt unit to survive

Chapter 5

Conclusion

In this research, I concentrated on learning effective tactics in RTS games. In my experiments, I use both evolutionary and co-evolutionary algorithms, which are techniques that were meant to search complicated spaces for vectors that maximize a fitness function. The solution space for my problem is a 24-dimensional space with many local maxima. The vectors in this space contain the parameters used to define potential fields between vehicles in an RTS game. The fitness function is defined as the score of the RTS game. I use the evolutionary algorithms to learn tactics against specific opponents, and the co-evolutionary algorithms to generalize tactics against a variety of opponents.

In the first phase of this work, I researched numerous methods for providing evolutionary learning. I settled on a hybrid algorithm, which I called Diversity Perserving Evolutionary Algorithm (DPEA), containing ideas from other authors' previous work, as well as ideas of my own. I found that my algorithm does a good job at finding a good set of tactics that had reasonable spatial diversity throughout the solution space. It did well at maintaining a spatially diverse population, something that the canonical GA does not do well. My algorithm was able to overcome many local maxima to find better solutions elsewhere in the solution space. Results showed that the tactics evolved using my algorithms were good at defeating known enemy players, but not as good at defeating enemies that have never been seen before.

In the second phase of research, I concentrated on finding better general tactics- those that would stand a better chance of defeating opponents that have never been seen before. For this phase, I combined my evolutionary algorithm with a co-evolutionary algorithm. I evolved two separate sets of tactics against each other, constantly changing the opponents that the tactics would have to defeat. Through this back-and-forth approach, I was able to evolve two sets of tactics that did well against

external enemies- those that have never been trained against. This shows that my algorithm for co-evolution can be used to generalize the learning of an evolutionary algorithm.

With these results, I can now expand the scope of this research to include other games and other strategies. Knowing that I have a basis for learning that balances generality with ability to learn, I can begin to explore more complex games with more complex rules. I can also look deeper at the theoretical foundations behind my algorithms in an attempt to fine-tune them to minimize learning time.

I am particularly interested in doing further research into the manner in which new chromosomes are generated from existing chromosomes. I believe that a certain amount of *feedback* could be utilized when selecting the combination of parent chromosomes, child chromosome generation algorithm, and parameters to use with the generation algorithm. My intuition is that making more informed choices in this area will reduce the amount of time taken to evolve chromosomes.

The idea of using potential fields as the only parameters into tactics was interesting to me for general work in evolutionary computing. However, it does not provide an optimal overall solution for a real RTS game. I would like to expand on this work as it relates to RTS games at a broader level. I believe that a higher-level AI can provide more input into the tactics. For instance, the tactics generated in this work will sometimes fail because they didn't learn *tie-breaking*. If a vehicle finds itself directly in the middle of two enemy vehicles, it will not move because the potential fields from each enemy vehicle will cancel each other. This is clearly not a good situation when the enemy units are stationary- none of the vehicles will want to move. While the potential field approach provided a good fitness function within the parameters of these experiments, potential fields alone may not be expressive enough to represent an optimal overall strategy for an RTS game.

Bibliography

[Aggarwal, et.al. 2001] C. Aggarwal, A. Hinneburg, D. Keim. *On the Surprising Behavior of Distance Metrics in High Dimensional Space*. Lecture Notes in Computer Science. Springer, 2001. pp 420-434

[Antonisse, 1989] Jim Antonisse. *A new interpretation of schema notation that overturns the binary encoding constraint*. Proceedings of the Third International Conference on Genetic Algorithms, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1989, pp 86-91.

[Avery, et.al. 2009] P. Avery, S. Louis, B. Avery. *Evolving coordinated spatial tactics for autonomous entities using influence maps*. IEEE Symposium on Computational Intelligence and Games, IEEE Press, 2009. pp 341-348

[Deb and Goldberg, 1989] K. Deb, D. Goldberg. *An investigation into Niche and Species Formation in Genetic Function Optimization*. Proceedings of the Third International Conference on Genetic Algorithms, Morgan Kaufmann, 1989, pp 42-50

[Eppstein, 1999] D. Eppstein. *Clustering* Spring 1999, Available: http://www.ics.uci.edu/~eppstein/280/cluster.html

[Eschelman, 1990] L. Eschelman. *The CHC Adaptive Search Algorithm: How to Have Safe Search When Engaging in Nontraditional Genetic Recombination*. Proceedings of the First Workshop on Foundations of Genetic Algorithms, Morgan Kaufmann, 1991, pp 265-283

[Fogel and Owens, 1966] L. Fogel, A. Owens, M. Walsh. *Artificial intelligence through simulated evolution.* Wiley, New York. 1966

[Fogel and Burgin, 1969] L. Fogel, G. Burgin. *Competitive goal-seeking through evolutionary programming.* Final Report, Contract AF 19(628)-5927, Air Force Cambridge Research Laboratories. 1969

[Fonseca and Fleming, 1993] C. Fonseca, P. Fleming. *Genetic Algorithms for Multiobjective Optimization: Formulation, Discussion and Generalization.* Proceedings of the Fifth International Conference on Genetic Algorithms, Morgan Kaufmann, 1993

[Hagelback and Johansson, 2008] J. Hagelback, S. Johansson. *A Multi-Agent Potential Field-Based Bot for a Full RTS Game Scenario.* IEEE Symposium On Computational Intelligence and Games, IEEE Press, 2008. pp 55-62

[Holland, 1975] J. Holland. *Adaptation in natural and artificial systems.* Ann Arbor, The University of Michigan Press, 1975

[Larose, 2006] D. Larose. *Data Mining Methods and Models*, Wiley-Interscience 2006, pp248-249

[Leigh, et.al. 2008] R. Leigh, T. Morelli, S. Louis, M. Nicolescu, C. Miles. *Finding Attack Strategies for Predator Swarms using Genetic Algorithms.* The 2005 IEEE Congress on Evolutionary Computation, IEEE Press, 2005 vol 3. pp 2422-2428

[Lucasius and Kateman, 1989] C. B. Lucasius and G. Kateman. *Application of genetic algorithms in chemometrics.* Proceedings of the Third International Conference on Genetic Algorithms, Morgan Kaufmann, 1989, pp 170-176.

[McCallum, et. al 2000] A McCallum, K. Nigam, L. Ungar. *Efficient clustering of high dimensional data sets with application to reference matching.* Proceedings of the 6th ACM SIGKDD, 2000, pp169-178

[Rosin and Belew, 1997] C. Rosin, R Belew. *New methods for competitive coevolution.* Cognitive Computer Science Research Group, UCSD, San Diego, CA, Tech. Rep. #CS96-491, 1997

[Russel and Norvig, 2003] S. Russell, P. Norvig. *Artificial Intelligence A Modern Approach.* Pearson Education, Inc. pp 111-116

[Sivanandam and Deepa, 2008] S. Sivanandam, S. Deepa. *Introduction to Genetic Algorithms*, Springer 2008, pp56

[Whitley, 1989] Darrell Whitley. *The GENITOR algorithm and selection pressure: why rank-based allocation of reproductive trials is best.* Proceedings of the Third International Conference on Genetic Algorithms, Morgan Kaufmann, 1989, 116-121.

[Wright, 1991] A. H. Wright. *Genetic Algorithms for Real Parameter Optimization.* Proceedings of the First Workshop on Foundations of Genetic Algorithms, Morgan Kaufmann, 1991, pp. 205-218.