Evolving Potential Fields to Direct Tactics in Real Time Strategy Games

Michael Oberberger, Sushil Louis, *Member, IEEE,* and Monica Nicolescu

***Abstract*— We investigate the evolution and co-evolution of tactics in real-time strategy games. In addition, we investigate the representation of tactics as combinations of potential fields thus pointing the way towards more compact representations for tactics that are also amenable to evolutionary search. Preliminary 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. Results from employing a co-evolutionary approach indicate that co-evolution can generate tactics that perform better over a set of opponents not previously encountered.**

I. INTRODUCTION

GENERATING tactics to implement strategies in RTS games is important for generating artificially intelligent opponents. Although there are many approaches to generating tactics in computer games, we use evolutionary computing approaches because we believe that evolutionary computing will yield flexible, adaptive tactics. Typically, researchers have built knowledge based systems leading to intelligent but predictable opponents. It would be interesting to provide new tactics that were less predictable, yet still effective.

We focus on using evolutionary algorithms to generate new tactics that are not based on any pre-conceived notions about how to play a game. We 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.

We focus on tactics in a game with one goal - eliminate the opponent. That is, it is not the goal to merely survive in the game – you must eliminate the other player to win and our evolutionary computing algorithm evolves tactics to accomplish this.

The game that we chose for use in our research involves a number of vehicles split between two teams. The goal of the game is for one team to eliminate the other team. The playing field contains no obstacles- this removed most of the complexity of path planning. It also allowed us to treat each vehicle in a game as if it were the generator of a potential field. It is by tuning the strengths of these potential fields that we derive our tactics. Other research in using potential fields has proven that they can be very effective in implementing tactics in RTS games [10]

Although our approach successfully evolves tactics that beat a particular opponent, we run up against the well-known problem in machine learning: over-specialization. We find that our evolved tactics over-specialize against the opponents used during evolution. In the second part of our work, we address this challenge by implementing a co-evolutionary algorithm that tends to generate more robust tactics. We show that these co-evolved tactics are better suited to defeat opponents that were never encountered during evolution compared to those tactics generated by evolution alone against specific opponents.

II. BACKGROUND

There are many known methodologies for defining a tactic for AI agents in Real-Time Strategy (RTS) games [10][3][13]. 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.

Finite State Machines

*FSM*s have been used frequently to determine how an agent will behave within a game [21]. An *FSM* is an efficient mechanism to give the appearance of intelligence in an agent. However, it does not have the ability to adapt itself beyond the logic that was used to design the *FSM* itself. An *FSM* controlled agent must have some concrete knowledge about what state it is in, what possible actions or observations it can take or make, and knowledge about what to do in response. Should a situation arise that the *FSM* was not designed to handle, that *FSM* controlled agent must

Manuscript received January 15, 2011. This work was supported in part by the U.S. Office of Naval Research under Award N00014-09-1-1121.

Michael Oberberger is with the Computer Science and Engineering Department, University of Nevada, Reno, 89502 USA (Phone: 775-830-0560; e-mail: mike@oberberger.com).

Sushil Louis is with the Computer Science and Engineering Department, University of Nevada, Reno, 89502 USA (Phone: 775-784-4315; e-mail: sushil@cse.unr.edu).

Monica Nicolescu is with the Computer Science and Engineering Department, University of Nevada, Reno, 89502 USA (Phone: 775-784-1687; e-mail: monica@cse.unr.edu).

revert to some default behavior. In order to account for a wide variety of situations, especially when the situations involve continuous-valued variables, the number of possible states grows rapidly, thus increasing the complexity of the *FSM*. Examples of *FSM* 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

Expert Systems

In other approaches, the game environment uses expert logic to implement tactics. While in some way similar to an *FSM*, Expert Systems typically are able to chain series of logical *rules* together to derive a *most advantageous action.*

By setting up a knowledge base about a particular game, an expert system could use the game's full current state to search the knowledge base for subsets of the state space that have been identified before. Using as much of the knowledge that is applicable (including the notion of probability in determining what is applicable) the expert system can deduce the most appropriate next action [22].

Evolutionary Systems

The topic of evolutionary algorithms has been studied since the 1960s [7]. They have been particularly successful at searching high-dimensional spaces for solutions to a variety of problems. We 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.

Other research in this area has used evolutionary algorithms to determine the strengths of simple potential fields, which were then used to derive *influence maps* [3]. The influence maps were then used to direct the movement of game entities.

Our 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, we show that these tactics can be effective in neutralizing an opponent player.

III. METHOD

The 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 other's space. Generally, an RTS game will not damage vehicles which collide, while in my game collisions are something that can severely penalize a player.

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.

A player loses the game once all of its vehicles are destroyed. The score of the game is used to determine the *Fitness Landscape* for the evolutionary algorithms. The score for a player is calculated using the following formula:

$$Score_{Winner} = \sqrt{\frac{RHP_{Winner} \cdot RVC_{Winner}}{HP_{Winner} \cdot VC_{Winner}}}$$

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

Where (R)HP are (Remaining/Starting) Hitpoints and (R)VC is (Remaining/Starting) Vehicle Count at the end of a game.

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 we did, we 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.

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 research, 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.

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).

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

$$F = cd^e$$

Where *d* is the distance between vehicles and *c* and *e* are parameters determined by the evolutionary algorithm. The three *virtual forces* which form our potential field can be loosely described as an *Attractor*, pulling close vehicles towards each other; a *Repulsor*, pushing close vehicles away from each other; and a *Spring*, which pulls vehicles that are far away closer together. For our experiments, the values for *c* and *e* are typified by the following table:

	Typical "*c*"	Typical "*e*"
Attractor	2,000	-2
Repulsor	500,000	-3
Spring	1.0	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:

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

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:

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

where the *x* values are those from the equation for *PF* for each of PF^0 and PF^1, respectively.

The encoding of the chromosome used for the evolutionary algorithm is a concatenation of two *PFS* vectors. The first *PFS* vector represents the potential field for *friendly* vehicles while the second *PFS* vector is the potential field of the *enemy* vehicles.

$$PFGrid = \{PFS_{Friend}, PFS_{Enemy}\}$$

For our experiments, we only used one type of vehicle in the game. This *PFGrid* would need to be expanded if there were other vehicle types in the game.

The *PFGrid* contains a total of 24 scalar floating point values. These values form the *chromosome* for the evolutionary algorithm. The chromosome represents a vector in 24-dimensional space. Each dimension contains the set of values represented by an 80-bit floating point number.

Since the elements 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, two vehicles that would otherwise not collide may now collide. This made us interested in defining a distance metric between two *PFGrids*.

To quote Aggarwal, "*... research results show that in high dimensional space, the concept of proximity, distance, or nearest neighbor, may not even be qualitatively meaningful*" [1]. In our experiments, we found that using the Euclidean Distance yielded similar numbers regardless of how different the tactics represented by the *PFGrids* were. Eschelman uses the Hamming distance to determine whether or not two chromosomes are close in the search space [6].

Knowing something about the search space, namely that two tactics can behave differently if any one of their PFGrid components are significantly different, we chose to use what we called a Floating Point Hamming Distance when calculating whether or not two *PFGrids* were close in the search space. After some of our initial work was completed,

but before we had any meaningful results, we had a sample database of roughly 40,000 *PFGrids*. We took a "snapshot" of these grids and created a vector of values based on the standard deviation for each value x_i in the *PF* vector. We called this vector the Standard Deviation Vector (*SDV*). We use it in calculating the Floating Point Hamming Distance. The calculation is straightforward:

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

where G^0 and G^1 were two *PFGrids*, *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 paper, *distance* or *spatial diversity* refers to this equation.

Evolution

We designed an algorithm called Diversity Preserving Evolutionary Algorithm (DPEA) [23]. We 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-valued alphabets [4]. By maintaining a higher diversity in the chromosomes, we were 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, we start with a population of randomly generated 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 is called the *initial epoch size* (*IES*) for the population.

An *epoch* is something that we 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 [19], we 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. Using one of these operations, we create a new chromosome from one or two parents. This new chromosome is then evaluated and a new fitness is assigned. The new chromosome is added to the epoch unless any of the following conditions is true:

• While evaluating the chromosome, one of the games played didn't complete in the allocated amount of time. In this case, the chromosome, when decoded, results in an invalid tactic.

• The chromosome is deemed spatially identical 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 essentially 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.

This set of operations is repeated until the current epoch is ready to become the next epoch. Generation of more epochs continues until we decided to stop it. Normally, we stopped evolution when we saw that there was no progress being made in the overall fitness of the current epoch after thousands of evaluations of new chromosomes.

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 [4][9], and is a key component of DPEA.

Every epoch begins with exactly *IES* chromosomes. In the case of epoch 0, these are randomly generated. 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.

Chromosomes will be continuously added to the epoch 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. We hypothesize about the implication of a large chromosome count for an epoch in [23].

We 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 [5][15], we found that this problem is quite a complex field of study unto its own. What we found was that all of the techniques available were computationally expensive. We wanted an algorithm that balanced good clustering with fast execution speeds.

We borrowed from the techniques that are well known, while reducing the iterative nature of the existing algorithms [23]. Essentially, we consider all of the chromosomes in the epoch with above average fitness. Starting at the best chromosome, we look for the set of all of the chromosomes that are far away spatially from this chromosome. As we 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, we add the set to the next epoch and repeat, using the next-highest fitness chromosome that still remains. We continue to do this until we have *IES* chromosomes forwarded to the new epoch.

The new epoch has a starting set of chromosomes that are all above average in fitness and are also spatially diverse.

Since this algorithm is our own method for niching [4], we were interested in how it would perform against some well known multi-modal solution spaces. We designed a test for this algorithm [23]. Empirically, our test runs show that our algorithm for epoch generation does indeed maintain multiple spatially diverse chromosomes that also have high fitness within the population.

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.

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. We use simple roulette wheel selection based on fitness.

Some chromosome generators require two parents. The second parent chromosome 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 chromosome from the first parent.

Chromosome Generation

In the canonical genetic algorithm, there are two main methods for new chromosome generation: crossover and mutation [11]. A canonical version of crossover is the Single-Point Crossover. Our 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 [20]. For instance, if the binary string represents four 8-bit integer numbers, there are four subsections of the 32 bits 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.

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 [20]. Since this can be done in a number of ways, we chose to use 5 different operations, each based in part on either the crossover or the mutation (or both) operations [23]. These are Uniform Mutation [12], N-Point Mutation, Uniform Crossover, N-Point Crossover, and Linear Interpolation [12][20].

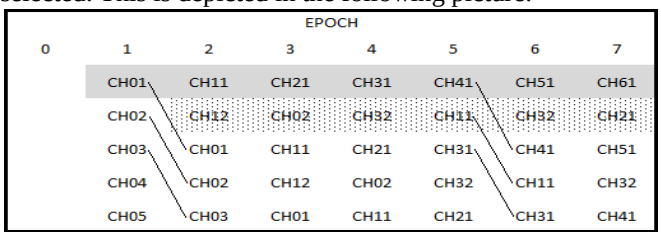
Co-Evolution

We found that the DPEA 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, we implemented a co-evolutionary algorithm, where two populations of chromosomes were evolved together and used each other to provide new opponents.

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 DPEA 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 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 the following picture:



In addition to these 5 opponents, we 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 used for evolution alone.

In evolution alone, the target fitness was determined by the initial chromosomes in only one population. However, in co-evolution, we 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, we wanted to see how the evolved tactics performed against opponents that had never been seen during evolution. To show this, we selected the best chromosome in each epoch from each population. For each of these chromosomes, we 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, we felt like we 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.

IV. Results / 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.

Scenarios

For all of the experimentation, we selected two game scenarios, each with six identical vehicles per player. The layouts of these scenarios were somewhat arbitrary. In the first scenario, we grouped player 1's vehicles into the two corners of the top half of the playing field while grouping all of player 2's vehicles in the middle of the lower half of the field. In the second scenario, we separated player 1's vehicles evenly across the top half of the field, while grouping player 2's vehicles in the bottom corners of the field. While we did want to have a variety of initial grouping configurations, 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).

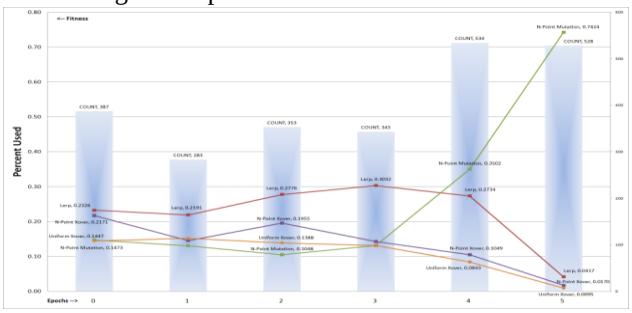
We also created two types of opponents. First, we used a *Static Agent*, where the desired speed and heading was simply set to zero. Second, we used a *Hand-Crafted* agent, which was simply a Herkermer agent whose chromosomes contained values that were manually set by us.

Evolution

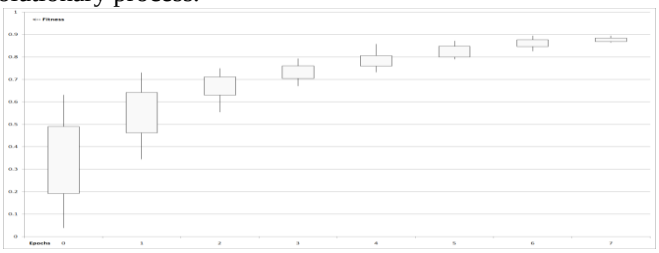
The process of evolution to improve tactics against a specific set of opponents went well. The overall fitness of the population improved over time in a manner typical of learning algorithms. It took the algorithm on average 2,000 evaluations to bring the fitness level to roughly 0.70.

While evolution did indeed increase the fitness of the population over time, an interesting phenomenon occurred in some of the evolution trials. Common to any search algorithm that is not exhaustive is the problem of local maxima/minima [17]. 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.

Consider the following picture. This graph shows 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. We 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 results show that our algorithm was adaptive to the current state in the search and was able to overcome many local maxima given enough time. It showed that 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. The preceding figure shows the range of fitness values for each epoch, with

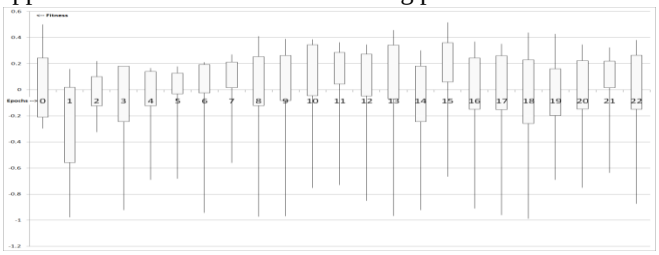
the range plus or minus one standard deviation above and below the average for each epoch highlighted. This population was evolved to a relatively high maximum fitness of 0.894.

However, in the graph following this we see 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 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. We expected this as the outcome for evolution conducted in this manner.

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. This was expected, as a 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 seen in the following picture:



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. We 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.

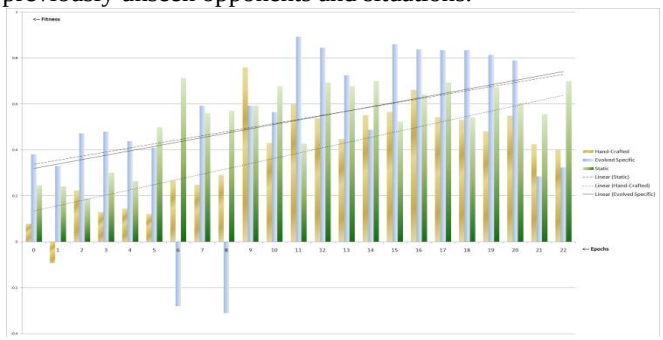
For the purpose of evaluating the tactics resulting from co-evolution, we selected a baseline against which the best chromosomes from each epoch could test themselves. This was accomplished by creating an environment that was not seen during co-evolution and evaluating the chromosomes from each epoch of co-evolution using this new environment. Using a scenario and three opponents that had not been used during co-evolution, we tested our approach. The opponents we chose are as follows:

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. We felt that changing the positions of stationary targets was sufficiently different to warrant including this opponent.

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

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.

The results of playing the tactics described by the best chromosomes from each epoch against the above described opponents are shown in the following graph. We are 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.



V. Conclusion

In this research, we concentrated on learning effective tactics in RTS games. In our experiments, we use both evolutionary and co-evolutionary algorithms. The fitness function is defined as the score of the RTS game. We 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, we researched numerous methods for providing evolutionary learning. We settled on a hybrid algorithm, which we called Diversity Preserving Evolutionary Algorithm (DPEA), containing ideas from other authors' previous work, as well as ideas of our own. We found that our 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. Our algorithm was able to overcome many local maxima to find better solutions elsewhere in the solution space. Results showed that the tactics evolved using our 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, we 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, we combined our evolutionary algorithm with a co-evolutionary algorithm. We 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, we were able to evolve two sets of tactics that did well against external enemies- those that have never been trained against. This shows that our algorithm for co-evolution can be used to generalize the learning of an evolutionary algorithm.

We are particularly interested in doing further research into the manner in which new chromosomes are generated from existing chromosomes. We 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. Our 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 us for general work in evolutionary computing. However, it does not provide an optimal overall solution for a real RTS game. We would like to expand on this work as it relates to RTS games at a broader level. We 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.

Videos of some of the results of our trials can be found at **http://tinyurl.com/4a5g4kl**

REFERENCES

[1] 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

[2] 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.

[3] 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

[4] 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

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

[6] 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

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

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

[9] C. Fonseca, P. Fleming. *Genetic Algorithms for Multiobjective Optimization: Formulation, Discussion and Generalization.* Proc. Fifth ICGA, Morgan Kaufmann, 1993

[10] 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

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

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

[13] 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

[14] 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.

[15] 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

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

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

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

[19] 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.

[20] 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.

[21] M. Buckland. *Programming Game AI by Example*. Jones & Bartlett Publishers. pp 43-82

[22] S. Russell, P. Norvig. *Artificial Intelligence A Modern Approach.* Pearson Education, Inc. pp 492-532

[23] M. Oberberger, "Evolving Potential Fields to Direct Tactics in Real Time Strategy Games", M.S. Thesis, ECSL, Univ. Nevada, Reno, 2010