Evolving Near Optimal Kiting Behavior with an Anchor Point Parameterization

Pavlos Androulakakis and Zachariah E. Fuchs

***Abstract*—In this paper, an evolutionary algorithm (EA) is used in conjunction with an *Anchor Point* parameterization to obtain near optimal kiting behavior for an Attacker in a 1v1 real-time combat scenario. In this scenario, an Attacker attempts to use its limited weapon energy to inflict as much damage as possible onto a fortified Defender while incurring as little damage as possible in return. An *Anchor Point* feedback controller parameterization allows the EA to represent the control boundaries more accurately and efficiently than our previously used grid based method. The evolved solution is compared with the analytically optimal solution to verify the EA's effectiveness and accuracy. The framework presented in this paper can be used to evolve a feedback controller for any unit in the 1v1 real-time combat scenario if its damage profile can be modeled as a function of distance.**

I. Introduction

There have been many significant contributions in the development of AI strategies for turn-based games such as chess [1] and go [2]. More recently, attention has shifted to games that can be played in real time. For this reason, real-time strategy (RTS) games have become a popular testing environment for modern game AI algorithms. The research in this area spans from academic test environments such as microRTS [3] and DeepRTS [4], to commercial games such as StarCraft [5].

In an RTS game, the player must use imperfect information to make a variety of decisions ranging from high-level tasks, such as base planning and army management, to low-level tasks, such as reconnaissance or combat between individual units. One popular method for handling such a large problem space is to decompose the tasks into separate individually solvable subproblems. For example, effective game playing agents have been developed for specific things such as build order [6], wall placement [7], army positioning [8], and even enemy resource prediction [9].

In this research, we closely examine one such subproblem: the micro-level 1v1 combat scenario. In this scope, the challenge for players and game developers alike becomes how to properly take advantage of the strengths and weaknesses of an individual unit in order for it to successfully defeat its opponent. The solution to this problem can vary widely based on the types and characteristics of the agents involved. For example, ranged units are typically weak in short-range combat but stronger at long distances. Alternatively, an assassin type unit may have a very strong short-range attack, but has low armor and is unable to survive extended 1v1 combat.

Pavlos Androulakakis is with the Department of Electrical Engineering, University of Cincinnati, Cincinnati, OH.

Zachariah E. Fuchs is with the Department of Electrical Engineering, University of Cincinnati, Cincinnati, OH.

The problem of individual unit micro-management has been examined and many novel solutions have been found. In [10], they use a hybrid approach in which an augmented Monte Carlo Tree Search (MCTS) algorithm is used to micro-manage units. In [8], they use a potential field method to direct each of the units in an army. In [11], the authors use a genetic algorithm to select the best actions for a combat unit from a library of preprogrammed actions.

In this work, we break from the traditional game AI methods and instead examine the problem from an optimal control perspective. Optimal control techniques are commonly used to develop optimal feedback control strategies for a wide range of complex control problems. Some examples include intercepting missiles [12], pursuit evasion problems [13], and defending high value targets [14], [15]. By examining the problem in this context, we can mathematically define the scenario and analytically compute the truly optimal control. This optimal solution can then be used as a benchmark with which to compare the performance of our proposed numerical solution.

In this paper, we present a controller representation and evolutionary framework that can be used to optimize a feedback control strategy for a unit of any given type in an 1v1 real-time combat scenario. For the purposes of demonstration, we specifically examine the case in which an *Attacker* with limited resources attempts to engage a stationary fortified *Defender*. The Attacker in this case is a melee unit that deals a high amount of damage at short range, but only has limited energy with which to execute its attack. Once the Attacker runs out of weapon energy, it is no longer able to inflict damage onto the Defender. The Defender is a melee unit that has fortified its position and is unable to move. It deals less overall damage, but has an infinite amount of energy with which to execute its attack. The goal of the Attacker is to inflict as much damage as possible onto the Defender while incurring as little damage as possible in return.

Our previous research [16] showed that an evolutionary algorithm can be used in conjunction with a grid parameterization to evolve an effective solution to this problem that exhibits what is commonly referred to as *kiting* behavior. While these results were successful, they were limited in that the resulting evolved feedback controllers were restricted to a grid. In order to more accurately represent the control boundaries, a high resolution grid needed to be used. As the grid resolution increased, the convergence time of the EA exponentially increased with it.

In this research, we expand upon our previous work and introduce an *Anchor Point* parameterization that will allow us to more accurately and efficiently represent the true underlying optimal solution. The anchor point method parameterizes a feedback controller by breaking the state space up into regions of discrete control. This idea was introduced in our previous paper [17] and is expanded upon here for use in evolving feedback controllers for the 1v1 combat problem.

Section II introduces the 1v1 real-time combat problem and defines the necessary state and utility equations. In Section III, we introduce the anchor point controller representation. In Section IV, we outline the evolutionary algorithm. The results of the EA are presented and compared with the true optimal solution in Section V. We conclude the paper with a summary of the results and discussion of future work in Section VI.

II. Problem Description

This problem consists of two agents; an Attacker and a Defender. The Attacker represents a mobile unit that can only carry a limited amount of weapon energy. The Attacker can choose which direction to move in as well as when to fire its weapon. The Defender represents a fortified unit that is able to constantly fire onto the Attacker with unlimited weapon energy. There are no control variables associated with the Defender as it is stationary and will implement the fixed strategy of constantly firing onto the Attacker. The goal of the Attacker is to use its limited weapon energy to inflict as much damage as possible onto the Defender while incurring as little damage as possible in return.

A. System Model

The state of the system is represented by a two dimensional state vector:

$$\mathbf{x} = \begin{bmatrix} d \\ \omega \end{bmatrix},$$

where d represents the distance between the two agents and ω represents the Attacker's remaining weapon energy. The dynamics of the system are

$$\dot{\mathbf{x}} = \begin{bmatrix} \dot{d} \\ \dot{\omega} \end{bmatrix} = \begin{bmatrix} v_{A} u_d \\ -r u_\omega \end{bmatrix}, \quad (1)$$

where v_{A} is the speed of the Attacker and r is the weapon energy depletion rate. The Attacker controls its motion with the control variable $u_d \in [-1, 1]$ and chooses whether or not to fire with the fire control variable, $u_\omega \in [0, 1]$.

Analysis of this problem using optimal control techniques has shown that the true optimal control has bang-bang behavior. This means that the control variables will only take on their min and max values unless constrained. In the case of motion control this means that $u_d \in \{-1, 1\}$ and in the case of fire control this means that $u_\omega \in \{0, 1\}$. The proof is not included in this paper due to space constraints.

B. Termination Conditions and State Boundaries

The skirmish ends when the Attacker retreats to a predefined retreat distance d_r. The terminal surface can be represented by all states that satisfy the equation $\Gamma(\mathbf{x}) = d - d_r = 0$.

Since the states d and ω represent physical aspects of our system, their values will be bounded by their corresponding physical limitations. In the case of distance, d, the state is lower bounded by 0 (collision) and upper bounded by d_r (retreat): $d \in [0, d_r]$. Similarly, in the case of weapon energy, ω, the state is lower bounded by 0 (empty) and upper bounded by its weapon energy capacity ω_c: $\omega \in [0, \omega_c]$.

C. Utility

The Attacker strives to inflict as much damage as possible onto the Defender while receiving as little damage as possible in return. Given an initial condition of $\mathbf{x}_0 := [d_0, \omega_0]$, the utility of an Attacker control strategy is defined as:

$$U(\mathbf{u}(\mathbf{x}); \mathbf{x}_0) = \int_{t_0}^{t_f} C_{A}(\mathbf{x}(t), \mathbf{u}(\mathbf{x}(t))) - C_{D}(\mathbf{x}(t)) dt, \quad (2)$$

where $\mathbf{u}(\mathbf{x}) = [u_d(\mathbf{x}), u_\omega(\mathbf{x})]$ is a vector containing the Attacker's feedback control variables, t_0 and t_f are the initial and final times, $\mathbf{x}(t)$ is the trajectory of the state computed with the dynamics shown in (1), and $C_{A}(\mathbf{x}, \mathbf{u}(\mathbf{x}))$ and $C_{D}(\mathbf{x})$ are the respective damage profiles of the Attacker and Defender. These damage profiles define the amount of instantaneous damage each player inflicts onto the other as a function of distance. In the general case, the damage profiles can be used to model any type of agent. For the purposes of this research, we model the scenario in which the Attacker and Defender deal high damage at short range with an exponential damage drop off as distance increases (melee units). Specifically, the Attacker is a specialized unit that deals extremely high damage at short range, but has a steep damage drop off with increasing distance. These damage profiles can be modeled with the equations shown below.

$$C_{A}(\mathbf{x}, \mathbf{u}(\mathbf{x})) = u_\omega(\mathbf{x}) \left(2e^{(-2d/5)} \right) \quad (3)$$

$$C_{D}(\mathbf{x}) = e^{(-d/5)} \quad (4)$$

Note that the Attacker's damage profile is multiplied by its fire control variable u_ω. When the Attacker is firing its weapon, $u_\omega = 1$, the damage profile follows the exponential curve. When the Attacker holds it fire, $u_\omega = 0$, the Attacker's damage profile will become 0. Figure 1 shows these two damage profiles for a range of distance values. The Attacker deals more instantaneous damage for all distances less than $d = 3.46$ and the the Defender deals more instantaneous damage for all distances greater than $d = 3.46$.

D. Optimization Problem Definition

The goal of this optimization problem is to find the Attacker control that will maximize the utility from all the initial conditions in the admissible state space. The overall utility of the Attacker U_{A} can be written as

$$U_{A}(\mathbf{u}(\mathbf{x})) = \frac{1}{d_r w_c} \int_0^{d_r} \int_0^{w_c} U(\mathbf{u}(\mathbf{x}); \mathbf{x_0}) dw_0 \, dd_0, \quad (5)$$

where d_r and ω_c are the upper bounds of the distance and weapon energy state respectively. This utility is an average of the Attacker's single point utility from all the admissible

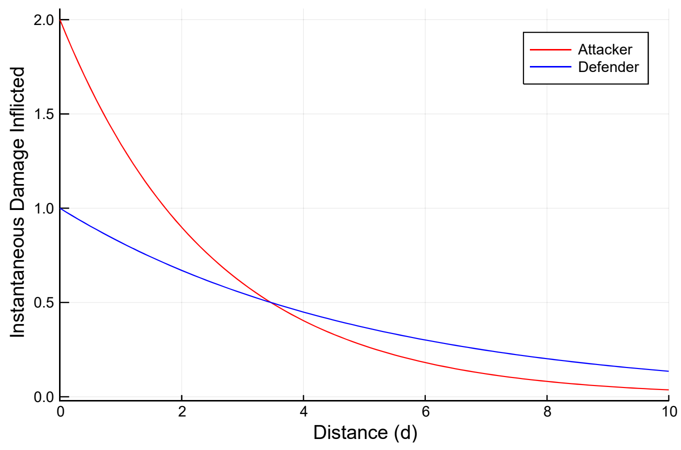

Fig. 1: Attacker and Defender Damage Profiles

initial conditions in the state space. With this overall utility equation, we define our optimization problem as

$$\max_{\mathbf{u}(\mathbf{x})} U_A(\mathbf{u}(\mathbf{x})). \tag{6}$$

The optimal Attacker control $\mathbf{u}^*(\mathbf{x})$ is the feedback controller that maximizes the overall Attacker utility function.

$$\mathbf{u}^*(\mathbf{x}) := \arg\max_{\mathbf{u}(\mathbf{x})} U_A(\mathbf{u}(\mathbf{x})). \tag{7}$$

III. Controller Representation

The goal of this research is to use an evolutionary algorithm to find the feedback controller, $\mathbf{u}^*(\mathbf{x};\mathbf{x}_0)$, that allows the Attacker to achieve maximum net damage in the 1v1 real-time combat problem. In order to evolve a feedback controller, we must first parameterize it so that we can apply the evolutionary operations of crossing and mutation to it.

Neural networks are a popular choice for parameterizing feedback controllers and there are many examples of research in which they are successfully integrated with an evolutionary algorithm [18] [19] [20] [21]. While these methods are quite successful, there is a high level of abstraction between the genotype (neural network) and the phenotype (resulting control output). For small neural networks, one can usually map a change in a particular gene to a corresponding change in the control output. However, as the neural networks increase in complexity (as is usually the case), it is not immediately clear how a change in a given gene will effect the output of the network. In the general case of developing effective feedback controllers, this is not a problem. However, a method with a more direct mapping between genotype and phenotype could assist researchers and AI developers to better understand their evolved solutions as well as allow them to develop custom crossing and mutation methods that are more useful for their particular problems.

For this reason, our past research has focused on introducing different types of controller representations that have a very clear mapping between genotype and phenotype. For example, in [16], we used a straight forward grid parameterization to evolve a feedback controller for this 1v1 real-time combat problem. The results of that research showed that our grid parameterization was effective in generating a usable feedback controller, but was not able to scale efficiently enough to represent complex control boundaries. In this research, we aim to improve upon those results by implementing an *Anchor Point* feedback controller parameterization.

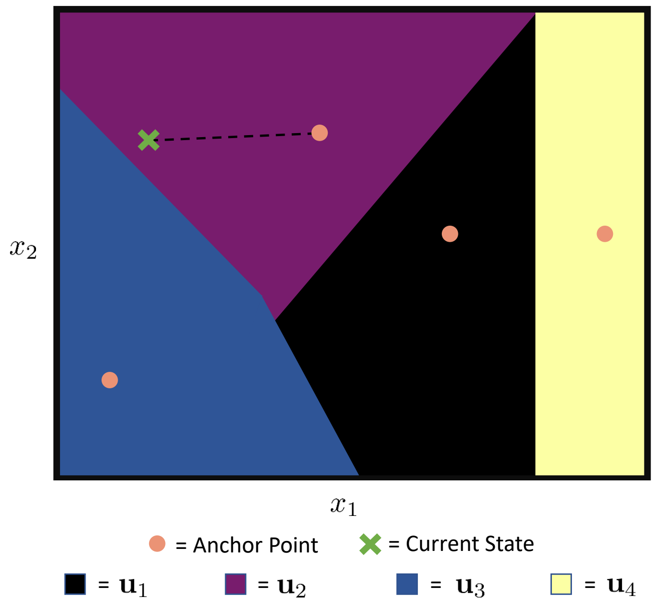

Fig. 2: Example of a 2D Anchor Point Parameterization

A. Anchor Point Parameterization

Anchor points were introduced in [17] as a method of partitioning the state space into regions of discrete control that can be utilized by a nearest neighbor switching controller (NNSC) in order to obtain a control output. In general, an anchor point $\mathbf{a} := [\mathbf{x}, \mathbf{u}]$ is defined as having a location in n dimensional space $\mathbf{x} = [x_1, x_2, \ldots x_n]$ and an associated control with m control variables $\mathbf{u} = [u_1, u_2, \ldots u_m]$. An anchor controller is then defined as the set of P anchor points $\mathbf{C} := \{\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_P\}$ used to partition the state space.

Figure 2 shows an example of a bounded 2D state space partitioned by four anchor points. The position of the anchor points in the state space is represented by the orange circles. The control output associated with each anchor point is represented by the surrounding color.

In the example shown, the control output at the current state (represented by the green $\times$) will be the control associated with the nearest anchor point (the purple control $\mathbf{u}_2$). In the most basic case, this nearest neighbor search can be done by finding the euclidean distance from the state to every anchor point in the controller and selecting the closest anchor point. This, however, is computationally inefficient and does not scale well as the number of anchor points increases. For this reason, we use a k-nearest neighbor (KNN) search algorithm to search a KD-Tree [22] constructed from the anchor points of the anchor controller. This is a relatively common practice for computationally efficient nearest neighbor searches and thus the details are not included in this paper.

B. Parameterizing the Attacker

In our problem, the state of the system is two-dimensional, $\mathbf{x} = [d, \omega]^T$. Therefore, each anchor point will also exist in the 2D state space. Additionally, since the Attacker has two control variables, u_d and u_ω, each anchor point will also have two control variables. Using this information, we define the anchor points in our 1v1 combat problem as $\mathbf{a} = [d, \omega, u_d, u_\omega]$. A candidate anchor controller is then defined as a set of P anchor points $\mathbf{C} = \{\mathbf{a}_1, \mathbf{a}_2, \dots, \mathbf{a}_P\}$.

Given a candidate anchor controller $\mathbf{C}$, the control output $\mathbf{u}(\mathbf{x}; \mathbf{C})$ at state $\mathbf{x} = [d, \omega]^T$ can by obtained by finding the nearest anchor point using the KNN search algorithm.

IV. Evolutionary Algorithm

The evolutionary algorithm presented in this section follows a canonical EA structure. By using a standard EA, we can focus our attention on the results of the controller representation. Once we are able to demonstrate our controller representation works with this standard EA, future work will aim to integrate our controller representation with state-of-the-art EA methods.

A. Overview

The EA begins by generating a population of N candidate controllers to serve as generation $G_0 := \{\mathbf{C}_1, \mathbf{C}_2, \dots, \mathbf{C}_N\}$. Each candidate controller is initialized with P anchor points that are randomly placed in the 2D bounded state space with random control outputs. This initial generation is pre-evaluated and then passed in as the first generation of the EA. Inside the EA, each generation is crossed, mutated, and evaluated using the methods described in the following subsections. The EA will continue to progress through the generations until a predefined number of generations M is reached. The agent with the highest fitness in the final generation is referred to as the *best evolved controller*.

B. Fitness Evaluation

The goal of this evolutionary algorithm is to evolve an Attacker that is able to inflict as much net damage as possible for any given initial condition within the admissible state space. In order to truly evaluate an agent's effectiveness at this task, we would have to test it from the entire continuum of states in the admissible state space as shown in equation (5). This is not computationally feasible. Instead, we approximate the true utility by sampling the state space at a $\gamma_1 \times \gamma_2$ grid of different initial conditions. We define this set of $n = \gamma_1\gamma_2$ initial conditions as $\mathbf{X}_0 = \{\mathbf{x}_{0,1}\ \mathbf{x}_{0,2}\ \dots\ \mathbf{x}_{0,n}\}$. These initial conditions can be represented as the set

$$\mathbf{X}_0 = \{(w, d) | \forall w \in \bar{w}, d \in \bar{d}\},$$

for

$$\bar{w} = \left\{ j\frac{w_c}{\gamma_1 - 1} \quad | \quad j \in \mathbb{Z} : 0 \le j \le \gamma_1 - 1 \right\}$$
$$\bar{d} = \left\{ j\frac{d_r}{\gamma_2 - 1} \quad | \quad j \in \mathbb{Z} : 0 \le j \le \gamma_2 - 1 \right\}.$$

Using these sample initial conditions, we generate an estimated utility function U_e as

$$U_e(\mathbf{u}(\mathbf{x}; \mathbf{C}); \mathbf{X}_0) := \frac{1}{n}\sum_{k=1}^{n} U(\mathbf{u}(\mathbf{x}; \mathbf{C}); \mathbf{x}_{0,k}). \tag{8}$$

The goal of the evolutionary algorithm then becomes to maximize this estimated utility. The higher the resolution of the $\gamma_1 \times \gamma_2$ initial condition grid, the closer we will be to approximating the true utility function U_A.

C. Crossing

In our anchor point representation, the order of the anchor points in the genome has no meaning. An anchor controller with anchor points $\mathbf{C} = \{\mathbf{a}_1, \mathbf{a}_2\}$ will have output identical to a controller $\mathbf{C} = \{\mathbf{a}_2, \mathbf{a}_1\}$ (same anchor points in a different order). For this reason, we decided to avoid traditional single or multi-point crossover in favor of a custom crossing technique that allows us to cross the agents in a more meaningful way.

The crossing process begins by randomly selecting two unique parents, $\mathbf{C}_1$ and $\mathbf{C}_2$, from the previous generation. Each agent has an equal chance of being selected as a parent. The child, $\mathbf{C}_c$, is then created using the following method.

First, two points (referred to as *split points*) are randomly placed somewhere in the 2D bounded state space; $s_1 = [d_1, \omega_1]$ and $s_2 = [d_2, \omega_2]$. Then, the parent's anchor points are assigned to the child based on their distance to the split points. Anchor points in parent $\mathbf{C}_1$ that are closer to split point s_1 than s_2 will be passed on to the child. Similarly, anchor points in parent $\mathbf{C}_2$ that are closer to split point s_2 than s_1 will be passed on to the child. Figure 3 shows an example of this crossing method.

By crossing the parents in this way, we are able to transfer anchor points in groups that influence entire regions of the state space. Since we are using two split points, this breaks the state space into two regions and is analogous to single-point crossover, but in the state space domain. If desired, the number of split points can be increased to achieve an effect similar to multi-point crossover. Note that this custom crossing technique is possible because our controller representation has a straightforward mapping between genotype and phenotype.

D. Mutation

Once crossing is complete, mutations are randomly applied to help explore the solution space and maintain diversity in the gene pool. In order to ensure that mutations do not degrade our best available solutions, the candidate controllers with fitness in the top 1% of the population (rounded up) are excluded from the mutation process. This practice is commonly referred to as *elitism*. Once the elite agents are removed from consideration, the remaining agents undergo the following two stages of mutation.

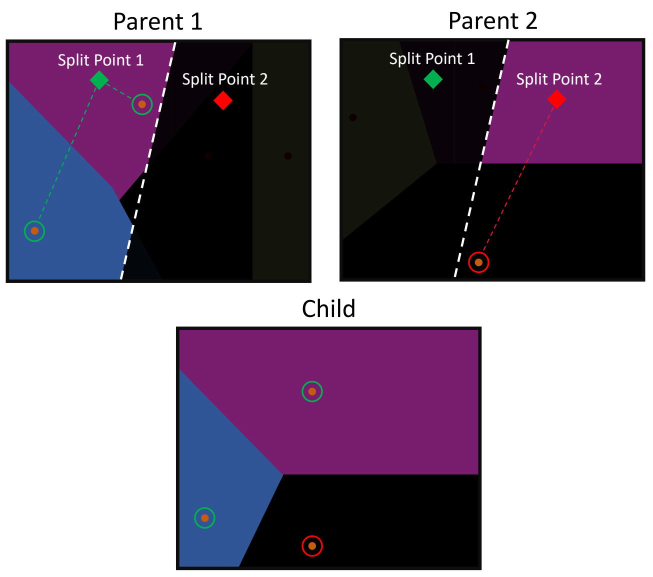

Fig. 3: Example of Crossing Two Parents Using Split Points

1) Minor Mutation: The first stage of mutation is referred to as *Minor* mutation. Minor mutations are designed to make small changes and occur at the individual anchor point level. First, each anchor point in each candidate controller has a $\mu_1\%$ chance of having its location mutated. For an anchor point at location (d, ω), the mutated anchor position (d_μ, ω_μ) can be computed as

$$\begin{aligned} d_\mu &= d + rand(-1, 1) \\ \omega_\mu &= \omega + rand(-1, 1) \end{aligned}$$

where $rand(-1, 1)$ is a uniform random number in the range -1 to 1.

Next, each anchor point has a $\mu_2\%$ chance of having its control value mutated. Since our control variables can only take on two discrete values, this mutation simply toggles the control value. In the case of the motion control, this will switch the control between -1 and 1. For the fire control, it will switch the value between 0 and 1.

2) Major Mutation: The second stage of mutation is referred to as *Major* mutation. Major mutations are designed to make significant genome alterations by changing the number of anchor points. These mutations happen on the controller level. Each candidate controller has a $\mu_3\%$ chance of having a new anchor point added to its controller. This new anchor point is randomly placed somewhere in the admissible state space with an appropriately selected random set of control values. Next, each candidate controller has a $\mu_4\%$ chance to have one of its anchor points randomly removed from its controller. This mutation is limited and cannot reduce the number of anchor points in a controller to less than 1.

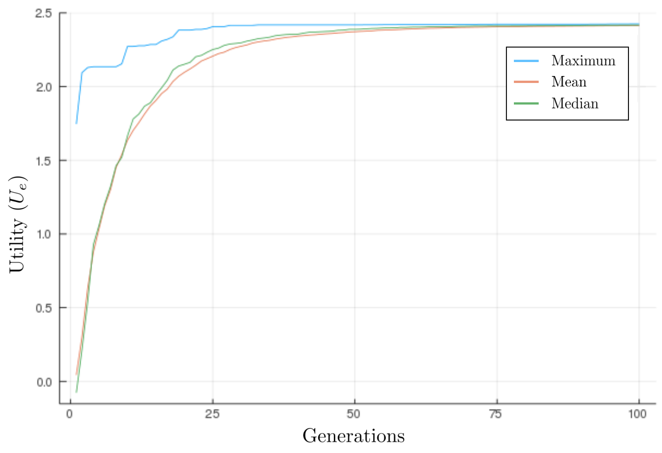

Fig. 4: Statistical Fitness Summary of the 100 EA Runs

V. Results and Analysis

The results presented in this section were generated using the following scenario and evolutionary parameters.

$$\begin{aligned} \text{Number of Generations} &: M = 100 \\ \text{Population Size} &: N = 36 \\ \text{Number of Initial Anchor Points} &: P = 10 \\ \text{Anchor Position Mutation Chance} &: \mu_1 = 5\% \\ \text{Anchor Control Mutation Chance} &: \mu_2 = 5\% \\ \text{Add Anchor Mutation Chance} &: \mu_3 = 5\% \\ \text{Remove Anchor Mutation Chance} &: \mu_4 = 5\% \\ \text{Utility Estimate IC Reolution} &: \gamma_1 = \gamma_2 = 21 \\ \text{Retreat Distance} &: d_r = 10 \\ \text{Weapon Energy Capacity} &: w_c = 10 \end{aligned}$$

These values were obtained through testing and are designed to best take advantage of the abilities of our computing resources. They represent one of many possible sets of evolutionary parameters that will yield successful results.

A. EA Statistical Results

Since evolutionary algorithms are stochastic, we must perform some form of statistical analysis in order to truly evaluate their performance. A single run of the evolutionary algorithm does not accurately represent the ability of the EA to find a solution. Therefore, we ran the evolutionary algorithm 100 times and recorded the fitness results over the course of each of the evolutions. Figure 4 shows the maximum (blue), average (red), and median (green) value of the best fitness in each generation over the course of the evolution for all 100 runs of the EA. This figure shows that in all 100 runs, the EA converges within 100 generations onto a solution with estimated utility U_e of approximately 2.42.

Figure 5 shows a composite image in which the best evolved solutions for each of the 100 EA runs are overlaid on top of each other. This image is a 2D representation of our state space (similar to the example in Figure 2) with distance on the x-axis and weapon energy on the y-axis. Black indicates a control

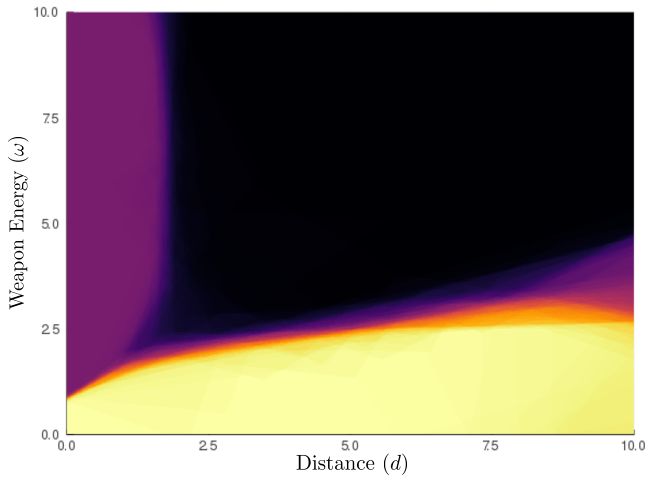

Fig. 5: Composite Image of Best Evolved Solutions

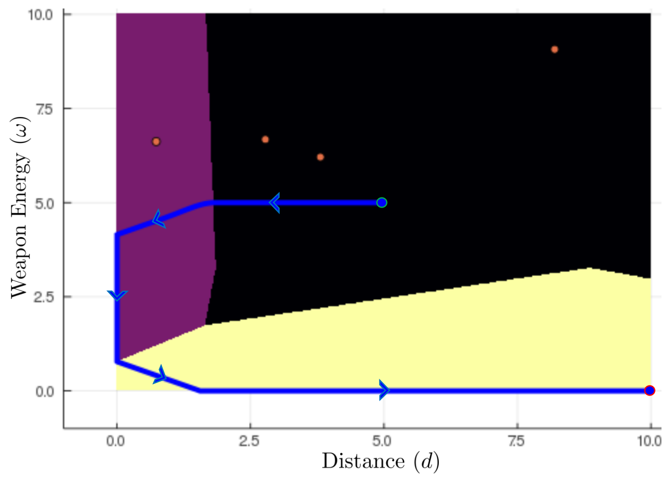

Fig. 6: Best Evolved Controller with an Example Kiting Trajectory (blue)

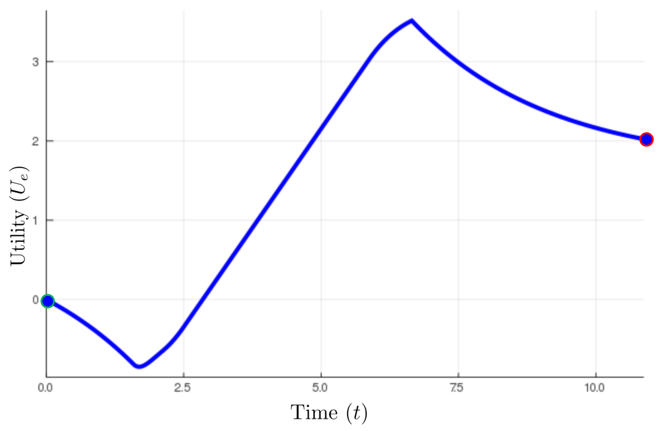

Fig. 7: Integral Utility Over the Course of the Trajectory Shown in Figure 5

of $\mathbf{u} = [-1, 0]$ (approach and hold fire), purple represents a control of $\mathbf{u} = [-1, 1]$ (approach and fire), and yellow represents a control of $\mathbf{u} = [1, 1]$ (fire and retreat). The anchor points that generated these state boundaries are not shown in the image for clarity, however they will be included in later individual analysis. This image shows that all 100 runs of the EA resulted in feedback controllers with state boundaries in approximately the same locations.

B. Best Evolved Controller

In this section, we examine the performance of the best evolved controller. For this analysis we used the EA run that resulted in the maximum fitness of all 100 runs in the previous statistical analysis. The anchor points of the best evolved controller are shown below.

$$\mathbf{C} = \left\{ \begin{array}{l} \mathbf{a}_1 = [0.72 \quad 6.56 \quad -1 \quad 1] \\ \mathbf{a}_2 = [2.78 \quad 6.67 \quad -1 \quad 0] \\ \mathbf{a}_3 = [3.80 \quad 6.21 \quad -1 \quad 0] \\ \mathbf{a}_4 = [8.17 \quad 9.01 \quad -1 \quad 0] \\ \mathbf{a}_5 = [4.12 \quad -1.65 \quad 1 \quad 1] \end{array} \right\}$$

Figure 6 shows the resulting state boundaries. The location of the anchor points is shown with the small orange dots. Note that while all anchor points were initialized in the bounded state space, they are not restricted to stay within those bounds. The minor mutations are able to gradually move the anchor points around. In this example, the EA found it beneficial to shift one of the anchor points, $\mathbf{a}_5$, to a position that is outside the range of this plot.

The blue line shows the path of an example scenario though the state space. In this example, the Attacker finds itself at an initial condition directly in the middle of the bounded state space (the green outlined circle at $\mathbf{x} = [5, 5]$). It begins by approaching the Defender while holding its fire. During this time, the Defender is constantly firing on the Attacker, causing the utility to decrease as is shown in Figure 7. Once the Attacker reaches a close enough distance ($d \approx 1.7$), it begins to fire its weapon while closing the remaining distance. This causes the utility to sharply increase as the Attacker is now dealing more damage than it is receiving. The Attacker then reaches the Defender and continues to use its weapon energy to inflict maximal damage onto the Defender. Once the Attacker's weapon energy begins to run low, it starts to retreat while expending whatever weapon energy it has left. The Attacker then runs out of weapon energy and continues to run away until it reaches the retreat distance (terminating at the red outlined circle at $d = 10$).

This example scenario shows what is commonly referred to as a kiting strategy. The Attacker held its limited weapon energy until it was able to reach a position in which it had a significant damage advantage, fired its weapon, and then retreated to safe distance once its weapon energy started to run out. This behavior was not presupposed onto the Attacker and is purely a result of the evolutionary algorithm.

If the solution to a problem with different types of agents is desired, all one would have to do is change the damage profiles C_A and C_D to model the new agents. Otherwise, the rest of the algorithm will remain unchanged. For this reason, we present this framework as a generalized method of finding a feedback controller for RTS units of any given type in a 1v1 combat scenario.

One clear limitation with this method is that it only considers the 1v1 combat scenario. Although combat in RTS games can frequently be broken down into a 1v1 scenario, many units are designed to work in small groups that perform better by compensating for each other's weaknesses. For example, a high damage unit supported by a healer, or a swarm of small weak units. Future work will aim to modify the framework presented in this paper to handle multi-agent scenarios.

C. *Comparison to Analytically Optimal Solution*

The analysis up to this point has qualitatively analyzed the solution and matches our intuition on what an optimal behavior may look like. However, without knowing the truly optimal solution, we cannot make any claims of optimality. In order to quantitatively evaluate the best evolved controller's performance, we must compare it to the true optimal solution. Since we framed the 1v1 real-time combat scenario in the context of an optimal control problem, we can analytically solve for the optimal solution using optimal control methods. However, it is important to note that calculating a closed-form analytic optimal solution is not practical or even feasible for every version of this problem. While this specific scenario can be analytically solved, other instances of this problem with different damage profiles may not. By validating our EA against the optimal solution, we increase our confidence in the optimality of the evolved solutions in situations where the optimal solution cannot be computed.

A comparison of the best evolved control boundaries to the truly optimal control boundaries is shown in Figure 8. The optimal control boundaries are shown as the overlaid bright green lines. This figure shows that the evolved boundaries closely match the general shape of optimal ones. For reference, the results of our previous research [16] are shown in Figure 9. These figures show that the anchor point method was able to represent the optimal control boundaries much more accurately than our previous grid method. Additionally, the anchor point method was able to approximate the optimal state boundaries with a 5×4 matrix (20 variables) whereas the grid method used a 15×15 matrix (225 variables). So not only was the anchor point method more accurate, but it was able to solve the problem with a representation that was 92% smaller.

Looking closely at Figure 8, one can see that there are still some noticeable imperfections in the boundaries produced by the anchor point representation. As stated in Section IV-B, the EA is optimizing an estimated utility function (8) instead of the true utility (5). If we plot the $\gamma_1 \times \gamma_2$ initial condition grid in the sate space (Figure 10), we can see that the overwhelming majority of the initial conditions are in regions of the state space that actually match the optimal solution. The ones that do not are circled in red. Overall, 98.41% (434 of 441) of our tested initial conditions result in solutions that match the optimal solution. If we compare the area in which the evolved solution matches the optimal solution (the entire continuum of untrained initial conditions) we find a similar success rate of 98.56%. This is an encouraging result as it shows the ability of the EA to produce a feedback controller that provides optimal results from nearly all initial conditions in the admissible state space. While this analysis doesn't guarantee the same performance for other unit types, it does increase our confidence in the optimality of other controllers evolved using this method.

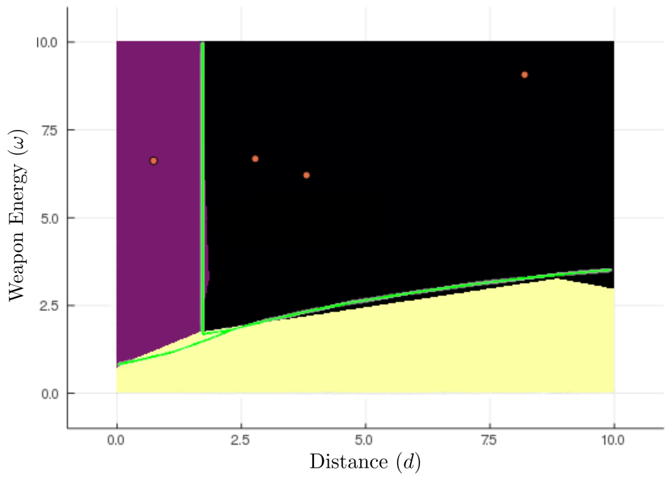

Fig. 8: Optimal State Boundaries (green) overlaid on Best Evolved Anchor Controller

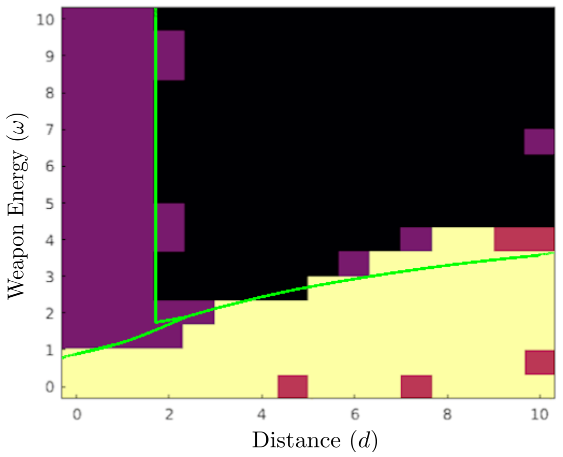

Fig. 9: Optimal State Boundaries (green) overlaid on Best Evolved Grid Controller

As close as the evolved solution is to the true optimal, we believe that it can be further improved. One method that is commonly used to improve the results of EAs is to follow the EA with a local search algorithm. Since EAs are global search algorithms, they are not very efficient at fine tuning their solutions. Instead, they are effective at searching the large solution space and converging on what is hopefully a peak containing the globally optimal solution. Future work can augment our method by applying a local search algorithm (such as gradient decent) to the best evolved solution to further fine tune the position of the anchor points.

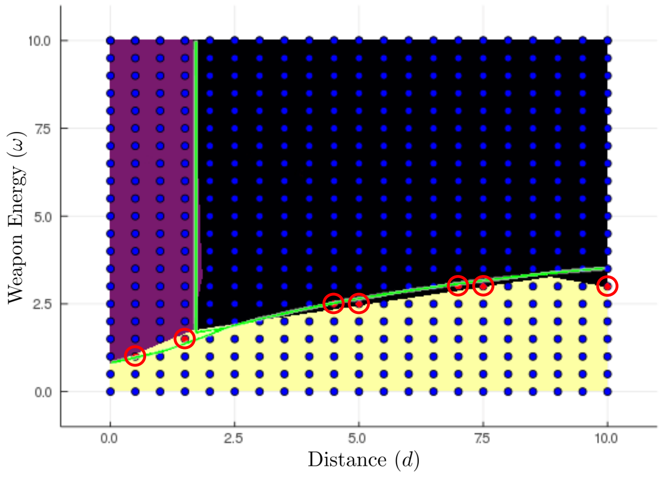

Fig. 10: Overlaid Tested Initial Conditions

VI. Conclusion

In conclusion, this research has demonstrated the effectiveness of an evolutionary anchor point method in obtaining a near optimal feedback controller for the 1v1 combat problem. By breaking down an RTS game into this basic 1v1 scenario, we were able to mathematically define the problem in an optimal control setting and compare the evolved results to the analytically optimal solution. The framework presented can be used to evolve a feedback controller for any given type of RTS unit in a 1v1 combat setting so long as its damage profile can be defined as a function of distance.

The anchor point parameterization method allowed us to encode a feedback controller in a much more accurate, compact, and scalable way than our previous grid representation. Comparing the numerical results to the analytically optimal solution showed that the EA was able to evolve an anchor point controller that matched the optimal solution 98.56% of the time.

Future work will aim to modify the presented evolutionary framework so that it can accommodate multi-agent scenarios. Additionally, we will attempt to integrate this anchor point method with state-of-the-art EA techniques and compare its performance to other popular feedback controller parameterizations such as neural networks.

References

[1] M. Campbell, A. Hoane Jr., and F. Hsu, “Deep blue,” *Artificial Intelligence*, pp. 57–83, 2002.

[2] D. Silver, J. Schrittwieser *et al.*, “Mastering the game of go without human knowledge,” *Nature*, pp. 354–359, 2017.

[3] S. Ontañón, “Informed monte carlo tree search for real-time strategy games,” in *2016 IEEE Conference on Computational Intelligence and Games (CIG)*, 2016.

[4] P. Andersen, M. Goodwin, and O. Granmo, “Deep rts: A game environment for deep reinforcement learning in real-time strategy games,” in *2018 IEEE Conference on Computational Intelligence and Games (CIG)*, 2018.

[5] S. Ontañón, G. Synnaeve *et al.*, “A survey of real-time strategy game ai research and competition in starcraft,” *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 5, no. 4, pp. 293–311, 2013.

[6] D. Churchill, M. Buro, and R. Kelly, “Robust continuous build-order optimization in starcraft,” in *2019 IEEE Conference on Games (CoG)*, 2019.

[7] M. L. M. Rooijackers and M. H. M. Winands, “Wall building in the game of starcraft with terrain considerations,” in *2018 IEEE Conference on Computational Intelligence and Games (CIG)*, 2018.

[8] T. Nguyen, K. Nguyen, and R. Thawonmas, “Potential flow for unit positioning during combat in starcraft,” in *2013 IEEE 2nd Global Conference on Consumer Electronics (GCCE)*, 2013.

[9] W. Hamilton and M. O. Shafiq, “Opponent resource prediction in starcraft using imperfect information,” in *2018 IEEE International Conference on Big Knowledge (ICBK)*, 2018, pp. 368–375.

[10] X. Neufeld, S. Mostaghim, and D. Perez-Liebana, “Evolving game state evaluation functions for a hybrid planning approach,” in *2019 IEEE Conference on Games (CoG)*, 2019.

[11] W. Hsu and Y. Chen, “Learning to select actions in starcraft with genetic algorithms,” in *2016 Conference on Technologies and Applications of Artificial Intelligence (TAAI)*, 2016, pp. 270–277.

[12] M. Pontani and B. A. Conway, “Optimal interception of evasive missile warheads: numerical solution of the differential game,” *Journal of Guidance, Control, and Dynamics*, vol. 31, no. 4, pp. 1111–1122, 2008.

[13] Z. E. Fuchs, P. P. Khargonekar, and J. Evers, “Cooperative defense within a single-pursuer, two-evader pursuit evasion differential game,” in *49th Conference on Decision and Control*, Dec. 2010, pp. 3091–3097.

[14] Z. E. Fuchs and P. P. Khargonekar, “An engage or retreat differential game with an escort region,” in *53rd Conference on Decision and Control*, 2014, pp. 4290–4297.

[15] ——, “Encouraging attacker retreat through defender cooperation,” in *50th Conference on Decision and Control and European Control Conference (CDC-ECC)*, Dec 2011, pp. 235–242.

[16] P. Androulakakis and Z. E. Fuchs, “Evolution of kiting behavior in a two player combat problem,” in *2019 IEEE Conference on Games (CoG)*, Aug 2019, pp. 1–8.

[17] P. Androulakakis and Z. E. Fuchs, “Evolutionary design of engagement strategies for turn-constrained agents,” in *IEEE Congress on Evolutionary Computation*, May 2017, pp. 2354–2363.

[18] K. Stanley and R. Miikkulainen, “Evolving neural networks through augmenting topologies,” *Evolutionnary Computation*, vol. 10, no. 2, pp. 99–172, 2002.

[19] S.-J. Han and S.-Y. Oh, “Evolutionary algorithm based neural network controller optimization for autonomous mobile robot navigation,” in *IEEE Congress on Evolutionary Computation*, vol. 1, May 2001, pp. 121–127.

[20] D. A. Miller, R. Arguello, and G. W. Greenwood, “Evolving artificial neural network structures: experimental results for biologically-inspired adaptive mutations,” in *IEEE Congress on Evolutionary Computation*, vol. 2, June 2004, pp. 2114–2119.

[21] N. Kohl and R. Miikkulainen, “Evolving neural networks for strategic decision-making problems,” in *Neural Networks, Special issue on Goal-Directed Neural Systems*, 2009.

[22] J. Bentley, “Multidimensional binary search trees used for associative searching,” in *Communications of the ACM*, September 1975.