Evolution of Kiting Behavior in a Two Player Combat Problem

Pavlos Androulakakis and Zachariah E. Fuchs

***Abstract*—We examine the use of an evolutionary algorithm to design a feedback controller for a two player combat problem. This problem consists of two players on a one-dimensional line. One player is referred to as the *Defender* and is held at the origin (unable to move). The other player is referred to as the *Attacker* and is free to move back and forth with a constant speed. The goal of the Attacker is to inflict as much damage as it can on the Defender while suffering as little damage as possible. The greater the difference between damage inflicted vs damage taken, the more successful the attack. The Attacker's controller is represented by a parameterized control vector. An evolutionary algorithm is then used to evolve populations of these control vectors in an attempt to obtain a near optimal feedback controller.**

I. Introduction

The development of cost effective engagement strategies is an important capability in many real-world and game scenarios. Targets are rarely defenseless, and the threat posed by their defensive capabilities represents some risk to the Attacker during an engagement. Additionally, the Attacker may not possess sufficient offensive capabilities to destroy the target in one pass. Instead, it must reengage the target over the course of multiple smaller skirmishes. Therefore, the Attacker must determine an optimal strategy for approaching and striking the target and then retreating to a safe distance while minimizing the cost and risk generated by the Target's defensive capabilities.

The general behavior in which an Attacker engages a Target and then retreats to a safe distance is known as *kiting* and is often utilized within real-time strategy games. There have been numerous efforts to utilize a variety of artificial intelligence methods to develop effective kiting behaviors for a wide range of games and engines. Some approaches utilize influence maps and similar approaches use potential fields [1], [2], [3], [4]. Most of these methods presuppose that kiting behavior is the desired behavior and then tune a controller or system parameters within the behavior to optimize for a particular scenario.

In this work, we do not impose any predefined behavior on the Attacker's strategy. Instead, we examine the problem from an optimal control perspective, in which we develop a feedback control strategy that calculates the optimal Attacker action based on the current state of the system. In the general case, the feedback controller is capable of modeling any arbitrary behavior, including kiting. There are many traditional control techniques for designing optimal controllers [5], [6]. Game-theoretic methods have also been utilized to calculate both strategic and tactical level engagement strategies and interactions [7], [8], [9], [10]. Although these methods calculate the true optimal solution for simple cases, there may not necessarily be analytical solutions for the resulting optimality conditions when the system contains nonlinear cost functions or dynamics. Instead, numerical methods must be used to optimize the controller or search the space of admissible control strategies.

Our previous work has shown that evolutionary algorithms can be used to develop optimized feedback controllers for a variety of scenarios [11], [12], [13]. In this paper, we examine a generalized system consisting of two agents: a static Defender and mobile Attacker. The system is represented in a two dimensional state space, and we use a system of ordinary differential equations to model system dynamics. This scenario can be found in many different games and game engines. For the scenarios considered in this paper, the Attacker strives to maneuver towards the Defender, fire its weapon, and then retreat to a predefined retreat boundary. The exact motion of the Attacker and when it fires its weapon is undefined a priori. We do not require the Attacker to retreat, but the Defender will continue to inflict cost onto the Attacker until it crosses the retreat boundary. The amount of damage inflicted onto the Defender is based on the integral of an instantaneous cost function dependent on the separation distance between the Defender and Attacker. Simultaneously, the Defender inflicts a cost onto the Attacker as a function of separation distance. The goal of the Attacker is to maximize the amount of damage inflicted onto the Defender while minimizing the amount of damage received.

We analyze the resulting optimal Attacker behavior produced by two different Attacker cost functions. The first cost function inflicts maximum damage at zero distance, which represents a melee type unit. The second Attacker cost function achieves maximum instantaneous damage at a nonzero distance from the Target, which models a type of ranged unit. As will be shown in the results, the evolutionary algorithm produces qualitatively different types of behaviors when engaging with different classes of units.

In Section II, we define the problem under consideration. We define the controller parameterization and evolutionary architecture used Section III. Numerical results are presented in Section IV, and we conclude the paper in Section V.

II. Problem Description

The scenario under consideration consists of two agents, an Attacker and a static Defender. The Defender represents

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

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

978-1-7281-1884-0/19/$31.00 ©2019 IEEE

a stationary high-value target such as a base, protected resource, or mission objective. The Defender uses its defensive capabilities to inflict damage (or a cost) onto the Attacker over the course of the encounter. The amount of damage inflicted by the Defender on the Attacker is a function of the separation distance between the agents. The Attacker is capable of maneuvering toward or away from the Defender and can inflict damage on the Defender by firing its weapon. Similar to the Defender, the amount of damage inflicted by the Attacker is a function of the separation distance. However, the Attacker possesses a finite amount of weapon energy, and once this energy is depleted, the Attacker can no longer inflict any damage on to the Defender. The Attacker strives to inflict as much damage as possible on the Defender while incurring minimal damage to itself.

The state of the system is represented by a two-dimensional state vector $\mathbf{x} = (d, w)$, where d represents the separation distance between the Attacker and Defender and w represents the Attacker's remaining weapon energy. The dynamics are modeled using a system of two ordinary differential equations:

$$\dot{d} = v_{A} u_1 \tag{1}$$
$$\dot{w} = r_{A} u_2, \tag{2}$$

where v_{A} represents the Attacker's speed and r_{A} represents the energy expenditure rate. The Attacker controls its motion through the control variable $u_1 \in [-1, 1]$, and it controls the activation of the weapon through $u_2 \in [0, 1]$. The Attacker's control vector is defined as $\mathbf{u} = (u_1, u_2)$.

The skirmish terminates when the Attacker retreats beyond the retreat boundary defined by d_r, which occurs when the state satisfies the termination condition:

$$\Gamma(\mathbf{x}) = d - d_r = 0. \tag{3}$$

We define the set of conditions that satisfy (3) as the terminal surface $\mathbf{S}_T := \{\mathbf{x} | \Gamma(\mathbf{x}) = 0\}$. The terminal time t_f is defined as the moment that the terminal condition is satisfied.

During the skirmish, the Defender constantly fires at the Attacker and inflicts damage onto the Attacker according the Defender cost function $C_{D}(\mathbf{x})$. The Attacker inflicts damage onto the Defender according the Attacker cost function $C_{A}(\mathbf{x}, \mathbf{u})$. The exact definitions of $C_{D}(\mathbf{x})$ and $C_{A}(\mathbf{x}, \mathbf{u})$ depend on the type of unit and its capabilities. In this paper, we examine how different types of cost functions generate optimal Attacker control strategies with qualitatively different behaviors.

The Attacker strives to inflict as much damage as possible, while incurring as little damage as possible from the Defender. This can be modeled by a utility function of the form

$$U_{A}(\mathbf{u}(t); \mathbf{x}_0) = \int_{t_0}^{t_f} C_{A}(\mathbf{x}(t), \mathbf{u}(t)) - C_{D}(\mathbf{x}(t)) dt, \tag{4}$$

where $\mathbf{x}_0$ is the state of the system at initial time t_0 and is defined as $\mathbf{x}_0 := \mathbf{x}(t_0) = (d_0, w_0)$. For an initial position $\mathbf{x}_0$, we can define an optimization problem to analytically calculate the optimal Attacker control strategy, $\mathbf{u}^*(t; \mathbf{x}_0)$, as

$$\mathbf{u}^*(t; \mathbf{x}_0) := \max_{\mathbf{u}(t)} U_{A}(\mathbf{u}(t); \mathbf{x}_0). \tag{5}$$

The resulting optimal solution can then be substituted into the dynamics (1)-(2), and integrated forward in time from $\mathbf{x}_0$ to obtain the corresponding optimal trajectory $\mathbf{x}^*(t, \mathbf{x}_0)$.

The resulting optimal control strategy $\mathbf{u}^*(t; \mathbf{x}_0)$ is only guaranteed to be optimal for the particular initial condition $\mathbf{x}_0$. If the skirmish is initialized at a different initial condition or if the Attacker would deviate from the resulting optimal trajectory $\mathbf{x}^*(t; \mathbf{x}_0)$, the optimization problem would need to be updated and solved from the new position. Although it may be possible to continually update and solve the optimal control problem in real-time, it would be more computationally efficient to precompute an optimal feedback controller $\mathbf{u}^*(\mathbf{x})$, in which the optimal control is a function of the current state instead of time. The overall performance of a feedback controller can be determined by averaging its utility over every state within the admissible state space:

$$U_{A}(\mathbf{u}(\mathbf{x})) = \frac{1}{d_r w_c} \int_0^{d_r} \int_0^{w_c} U_f(\mathbf{u}(\mathbf{x}); \mathbf{x_0}) dw dd, \tag{6}$$

where w_c is the maximum weapon energy and $U_f(\mathbf{u}(\mathbf{x}); \mathbf{x_0})$ represents the resulting utility of the feedback controller initialized at the given state $\mathbf{x_0} = (d, w)$

$$U_f(\mathbf{u}(\mathbf{x}); \mathbf{x_0}) = \int_{t_0}^{t_f} C_{A}(\mathbf{x}(t), \mathbf{u}(\mathbf{x}(t))) - C_{D}(\mathbf{x}(t)) dt. \tag{7}$$

Using the overall utility function (6), we can pose an optimization problem in which we optimize the performance of the feedback controller over the entire state space:

$$\max_{\mathbf{u}(\mathbf{x})} U_A(\mathbf{u}(\mathbf{x})). \tag{8}$$

In this paper, we will attempt to solve this optimization problem by using an evolutionary algorithm to evolve a discretized parameterization of the feedback controller $\mathbf{u}(\mathbf{x})$.

III. Evolutionary Architecture

In order to evolve a feedback controller, we must define a parametric representation with a set of tunable parameters. Once parameterized, we can use an evolutionary algorithm (EA) to evolve populations of these feedback controllers through the processes of crossing and mutation.

A. Controller Encoding

We begin by defining the range of all possible states. The relative distance is bounded between 0 (collocation) and d_r (retreat). The weapon energy is lower bounded by 0 (empty weapon energy) and upper bounded by w_c (maximum weapon energy capacity). Together, these bounds define the set of admissible states

$$\mathbf{x} \in \{0 < d < d_r, 0 < w < w_c\}.$$

As defined in Section II, the Attacker's feedback control strategy, $\mathbf{u}(\mathbf{x})$, consists of two separate control functions: $u_1(\mathbf{x})$ and $u_2(\mathbf{x})$. The motion control function, $u_1(\mathbf{x})$, can only take on the discrete values of $u_1 \in \{-1, 1\}$ where -1 corresponds to *move left* and 1 corresponds to *move right*. The fire control function, $u_2(\mathbf{x})$, is also discrete valued

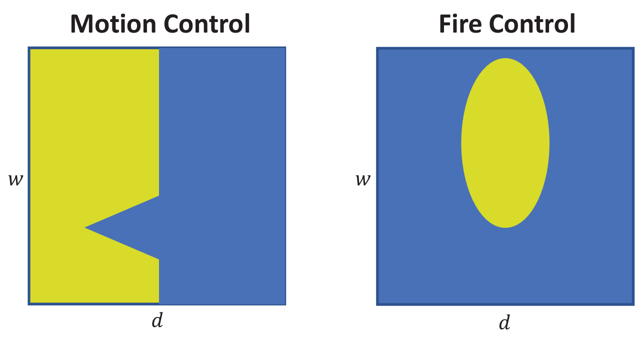


Fig. 1: 2D State Space Control Example

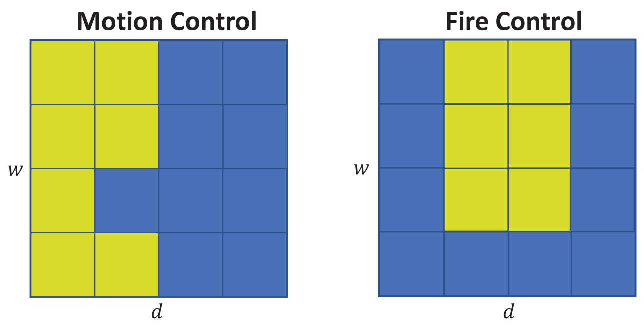


Fig. 2: Grid Parameterization Example

with $u_2 \in \{0, 1\}$, where 0 corresponds to *hold fire* and 1 corresponds to *fire*. By bounding the state and discretizing our control output in this way, we can represent the feedback controller as two separate two-dimensional state spaces. Figure 1 shows an example of two such state spaces with arbitrary boundaries defining regions of control. The color of the region indicates the value of the controller output at that given state. Yellow represents a control output of $u_1 = 1$ for motion control and $u_2 = 1$ for fire control. Blue represents a control output of $u_1 = -1$ for the motion control and $u_2 = 0$ for the fire control.

In order to complete the parameterization of the controller, we must be able to represent the boundaries between the regions of control. This can be done with many different methods such as neural networks, support vector machines, or anchor points [14], [11]. While these methods can be very effective, they can also add more complexity than is necessary in certain problems. For this reason, we aim to use a simpler controller representation: a parameterized grid structure. The grid resolution is defined by parameters ϵ_1 and ϵ_2. The higher the grid resolution, the more complex boundaries this parameterization is able to approximate. Figure 2 shows a parameterized example of the state space boundaries from Figure 1 for $\epsilon_1 = \epsilon_2 = 4$.

By parameterizing our controllers in this way, we can represent a candidate control strategy $\mathbf{c}$ as a set of two-dimensional matrices:

$$\mathbf{c} = \{\mathbf{m}, \mathbf{n}\},$$

where

$$\mathbf{m} = \begin{pmatrix} m_{1,1} & \dots & m_{1,\epsilon_2} \\ \vdots & & \vdots \\ m_{\epsilon_1,1} & \dots & m_{\epsilon_1,\epsilon_2} \end{pmatrix} \; \mathbf{n} = \begin{pmatrix} n_{1,1} & \dots & n_{1,\epsilon_2} \\ \vdots & & \vdots \\ n_{\epsilon_1,1} & \dots & n_{\epsilon_1,\epsilon_2} \end{pmatrix}.$$

A feedback controller can then be parameterized in terms of $\mathbf{c}$ as $\mathbf{u}(\mathbf{x}; \mathbf{c}) = (u_1(\mathbf{x}; \mathbf{c}), u_2(\mathbf{x}; \mathbf{c}))$, where u_1 and u_2 are defined as

$$\begin{aligned} u_1(\mathbf{x}; \mathbf{c}) &= \mathbf{m}_{i,j} \\ u_2(\mathbf{x}; \mathbf{c}) &= \mathbf{n}_{i,j} \end{aligned} \quad ,$$

for

$$\begin{aligned} i &= floor(\frac{\epsilon_1 d}{d_r}) + 1 \\ j &= floor(\frac{\epsilon_2 w}{w_c}) + 1 \end{aligned}$$

where ϵ_1 and ϵ_2 are the resolution of the grid parameterization and d_r and w_c are the given boundaries of the state variables.

B. Initial Population

We begin the evolutionary algorithm with a population of M candidate controllers to serve as our first generation $G_0 = \{\mathbf{c}_1, \mathbf{c}_2, ..., \mathbf{c}_M\}$. These candidate controllers are initialized with random control values uniformly drawn from the set $\{-1, 1\}$ for motion control, and $\{0, 1\}$ for fire control.

After generating the initial population, G_0, the evolutionary algorithm will fill the next generation, G_1, as follows. The controllers with fitness in the top 5% of the old generation are considered *elite* and are passed on to the next generation without any change. This ensures that the top performing controllers persist through to the next generation and don't accidentally get degraded by the crossing and mutation operations. The remaining 95% of the next generation is filled with the results of crossing and mutation. This process will repeat until the evolution reaches a predefined number of generations and ends with generation G_f.

C. Fitness Evaluation

As shown in (6), the overall utility of a candidate feedback controller is obtained by evaluating it from a continuum of initial conditions across the admissible state space. Since this is computationally infeasible to do, we approximate the overall utility by sampling the state space at a $\gamma_1 \times \gamma_2$ grid of different initial conditions. We define the 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

$$\begin{aligned} \bar{w} &= \left\{ j\frac{w_c}{\gamma_1} \quad | \quad j \in \mathbb{Z} : 1 \leq j \leq \gamma_1 \right\} \\ \bar{d} &= \left\{ j\frac{d_r}{\gamma_2} \quad | \quad j \in \mathbb{Z} : 1 \leq j \leq \gamma_2 \right\}. \end{aligned}$$

An individual candidate controller $\mathbf{c}_i$ can be used to parametrize a feedback controller $\mathbf{u} = (\mathbf{x}, \mathbf{c}_i)$ which is then

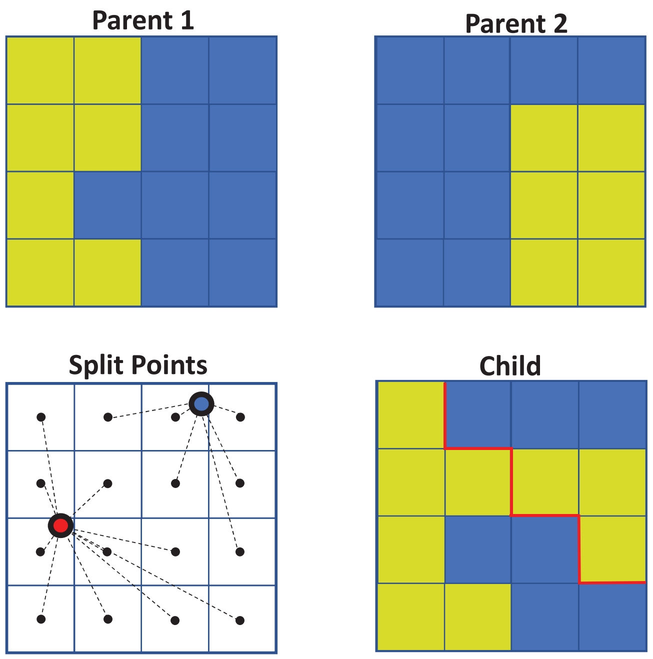


Fig. 3: Crossing Example

evaluated individually at each of these initial conditions to return a utility as described in (7). The total fitness of a candidate controller $\mathbf{c}_i$ is defined as the average of all the utilities resulting from evaluating $\mathbf{u} = (\mathbf{x}, \mathbf{c}_i)$ at the initial conditions in the set $\mathbf{X}_0$ and is defined as

$$f_i(\mathbf{u}(\mathbf{x}; \mathbf{c}_i); \mathbf{X}_0) := \frac{1}{n} \sum_{k=1}^{n} U_f(\mathbf{u}(\mathbf{x}; \mathbf{c}_i); \mathbf{x}_{0,k}).$$

This total fitness is used throughout the rest of the evolutionary algorithm to judge how fit the candidate controller is with respect to the rest of the population.

D. *Crossing*

Crossing begins by randomly selecting two parents from the previous generation. The likelihood that a candidate controller is selected for crossing is proportional to its relative fitness with respect to the rest of the population. The more fit the individual, the higher the probability that it will be selected for crossing.

Once two parents, (referred to as $\mathbf{c}_a$ and $\mathbf{c}_b$), are selected, a child controller $\mathbf{c}_c$ is created by subdividing the parent's state space using a *split point* method. A *split point* is defined as a point that is randomly selected from the admissible bounded state space with uniform probability distribution. For this crossing method, two split points are generated; $s_1 = \{d_1, w_1\}$ and $s_2 = \{d_2, w_2\}$. Given these two split points, the value of a cell in the child's control matrix (take for example the motion control matrix $\mathbf{m}_{c_{i,j}}$) can be defined as

$$\mathbf{m}_{c_{i,j}} = \begin{cases} \mathbf{m}_{a_{i,j}} & \ell_1 < \ell_2 \\ \mathbf{m}_{b_{i,j}} & \ell_1 \geq \ell_2 \end{cases},$$

for

$$\ell_1 = \sqrt{(d_i - d_1)^2 + (w_j - w_1)^2}$$
$$\ell_2 = \sqrt{(d_i - d_2)^2 + (w_j - w_2)^2},$$

where $\{d_i, w_j\}$ is the state value of the center of grid cell $\mathbf{m}_{c_{i,j}}$:

$$d_i = (i-1)\frac{d_r}{\epsilon_1} + \frac{d_r}{2\epsilon_1}$$
$$w_j = (j-1)\frac{w_c}{\epsilon_2} + \frac{w_c}{2\epsilon_2}.$$

This process computes the distance from each split point to the center of every grid cell. Grid cells closer to split point s_1 will have their control assigned from the corresponding grid cell in parent $\mathbf{c}_a$. Grid cells closer to split point s_2 will have their control assigned from the corresponding grid cell in parent $\mathbf{c}_b$. An example of this process for the motion control matrix is show in Figure 3. The red dot indicates split point s_1 and the blue dot indicates split point s_2.

This process is performed independently for each of the control matrices; once for the motion control matrix, and once for the fire control matrix. The result is a child whose control matrices contain information from both parents. Since this crossing is fitness based, candidate controllers with higher overall fitness will be more likely to produce children and pass on their control information.

E. *Mutation*

After a child $\mathbf{c}_c = \{\mathbf{m}, \mathbf{n}\}$ is produced from crossing, random mutations are applied to create a mutated version of the child $\bar{\mathbf{c}}_c = \{\bar{\mathbf{m}}, \bar{\mathbf{n}}\}$. Each cell in each of the child's control matrices has a μ chance of being mutated. When a mutation occurs, the cell's control value is randomly reassigned from the admissible control values. For a cell in the mutated motion control matrix $\bar{\mathbf{m}}_{i,j}$, the resulting distribution of control values is

$$p(\bar{\mathbf{m}}_{i,j}) = \begin{cases} 1-\mu & \bar{\mathbf{m}}_{i,j} = \mathbf{m}_{i,j} \\ \mu & \bar{\mathbf{m}}_{i,j} \in \{-1, 1\} \end{cases}.$$

For the fire control matrix, the resulting distribution for the control values is

$$p(\bar{\mathbf{n}}_{i,j}) = \begin{cases} 1-\mu & \bar{\mathbf{n}}_{i,j} = \mathbf{n}_{i,j} \\ \mu & \bar{\mathbf{n}}_{i,j} \in \{0, 1\} \end{cases}.$$

The resulting mutated child $\bar{\mathbf{c}}$ is then added to the set of candidate controllers for the next generation.

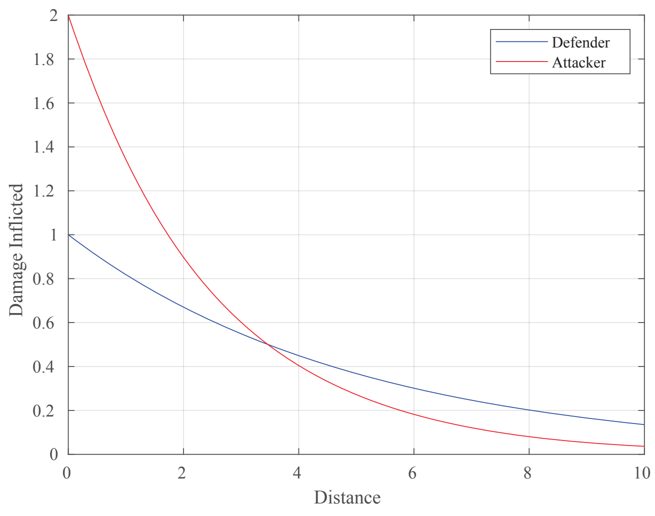


Fig. 4: Cost Function of Melee Attacker vs Melee Defender

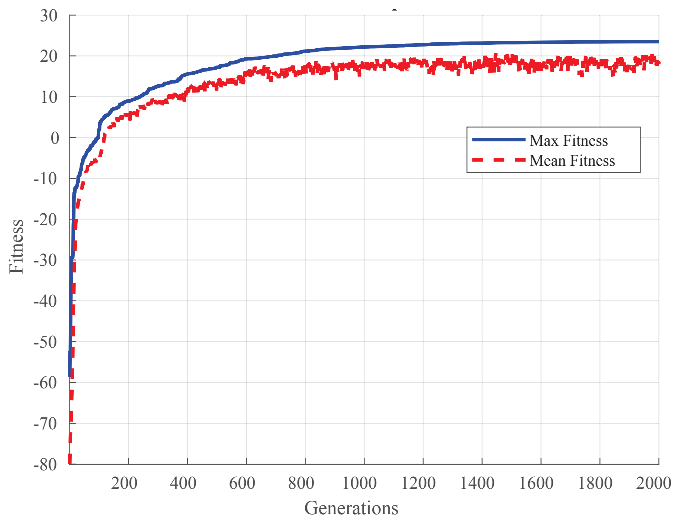


Fig. 5: Fitness Over the Generations

IV. Results and Analysis

The results in this section were obtained by evolving a population of 200 candidate controllers over 2000 generations with the following parameters:

Attacker Velocity:	$v_{\scriptscriptstyle A} = 0.2$
Weapon Energy Depletion Rate:	$r_{\scriptscriptstyle A} = 0.1$
Retreat Distance:	$d_r = 10$
Maximum Weapon Energy :	$w_c = 10$
Controller Grid Resolution:	$\epsilon_1 = \epsilon_2 = 16$
Initial Condition Resolution:	$\gamma_1 = \gamma_2 = 16$
Mutation Chance:	$\mu = 0.2\%$

A. Melee Attacker vs. Melee Defender

We begin with the case in which both Attacker and Defender have damage profiles that correspond to close range melee units. The Defender is a balanced unit that inflicts moderate damage up close and has a slow exponential damage drop off as the relative distance increases.

$$C_{\scriptscriptstyle D}(\mathbf{x}) = e^{(-d/5)} \tag{9}$$

The Attacker is a specialized unit that inflicts high damage at close range, but as a consequence has a fast exponential drop off as distance increases.

$$C_{\scriptscriptstyle A}(\mathbf{x}, \mathbf{u}) = u_2 \left(2e^{(-2d/5)}\right) \tag{10}$$

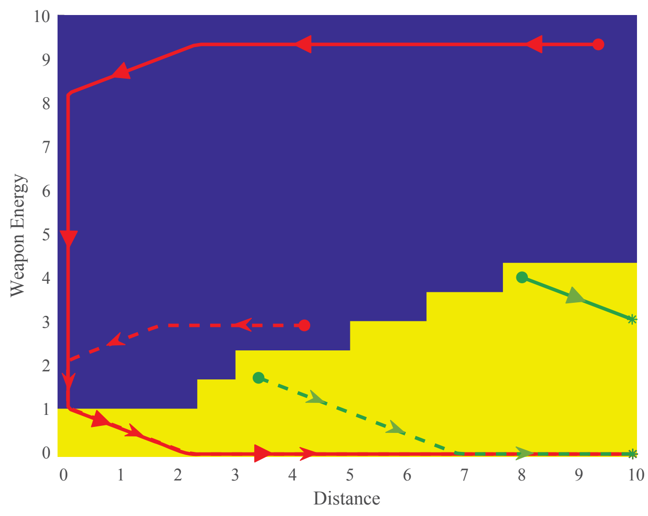
Figure 4 shows both of these damage profiles overlaid on each other for $u_2 = 1$. For all distances closer than $d = 3.46$, the Attacker has a damage advantage. For all distances greater than $d = 3.46$, the Defender has a damage advantage.

The evolutionary algorithm was run and the resulting fitness change over the 2000 generations is shown in Figure 5. The maximum fitness follows a logarithmic shape, with most of the improvements happening early in the evolution. We can also see that at no point does the maximum fitness decrease from one generation to the next. This is a direct result of elitism. The candidate controller with the highest fitness in the last generation is considered the best evolved controller and is examined in the following analysis.

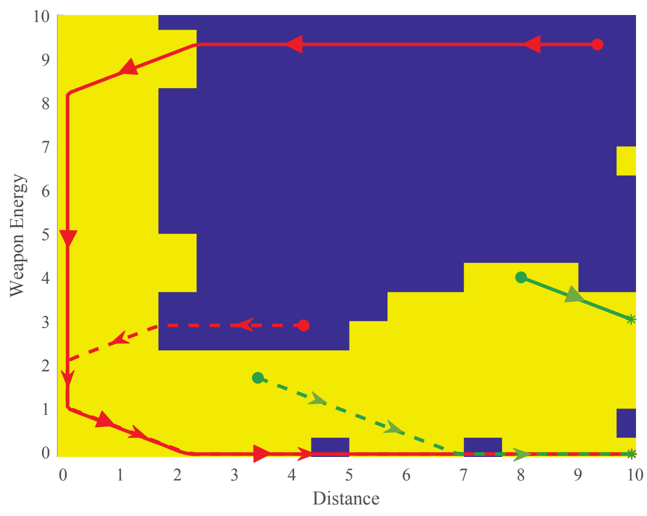
Figure 6a shows the motion control matrix and Figure 6b shows the fire control matrix of this best evolved controller. Four sample trajectories are shown by the solid/dashed red/green overlaid lines. A solid line indicates a trajectory that is initialized from a state within the EA training set ($\mathbf{x}_0 \in \mathbf{X}_0$). A dashed line indicates a trajectory that is initialized from an untrained initial condition ($\mathbf{x}_0 \notin \mathbf{X}_0$). The color of the line indicates the type of strategy implemented. A red line indicates an *kiting* strategy and a green line indicates a *retreat* strategy.

We begin by looking at the solid red line. This trajectory starts in a trained initial condition with close to full weapon energy at a far distance, $x_0 = \{9.33, 9.33\}$. We can see that from this initial condition, the Attacker starts by holding its fire while approaching the Defender. Once the Attacker gets close, it begins to expend its weapon energy and inflict damage onto the Defender. It remains in this position until its weapon energy starts to run out. At this point the Attacker begins to retreat while continuing to inflict damage until it completely runs out of weapon energy.

The integrated utility over the course of this trajectory can be seen as the solid red line in Figure 7. We can see that as the Attacker is approaching, it suffers an increasing amount of damage from the Defender. It seems counterintuitive for the Attacker to hold its fire since it could be reducing its net damage by inflicting damage on the Defender. The reason for this behavior is the finite weapon energy of the Attacker. By holding fire, the Attacker is able to wait until it has a significant damage advantage before it begins attacking. This allows the Attacker to get as much net damage as possible from its limited energy. At time $t = 35$ we can see that the Attacker begins attacking the Defender and the utility starts to sharply increase. This result shows that the evolutionary algorithm was able to develop a control strategy to conserve the limited weapon energy of the Attacker and only use it



(a) Motion Control



(b) Fire Control

Fig. 6: Best Evolved Melee Attacker Grid Controllers

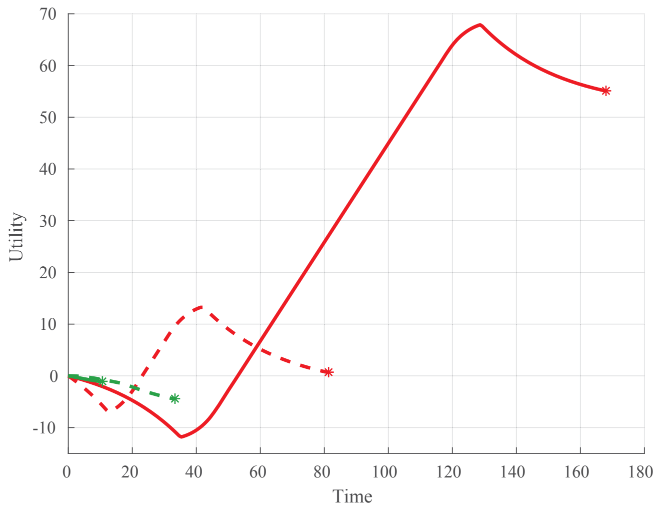


Fig. 7: Integral Utility of Trajectories

when the net damage is the highest.

The dashed red line shows a trajectory that starts from the untrained initial condition of $x_0 = \{4.2, 2.9\}$. We can see that this trajectory implements a very similar strategy to the red solid line. It engages the Defender, attacks from a close distance, and then disengages The integral utility for the trajectory is shown as the dashed red line in Figure 7. Even though this Attacker started from an untrained initial condition, it was able to implement the kiting strategy and inflicted more damage on the enemy than it received in turn. This type of result indicates that the evolved solution is generalized enough to translate its performance into initial conditions it has never seen before.

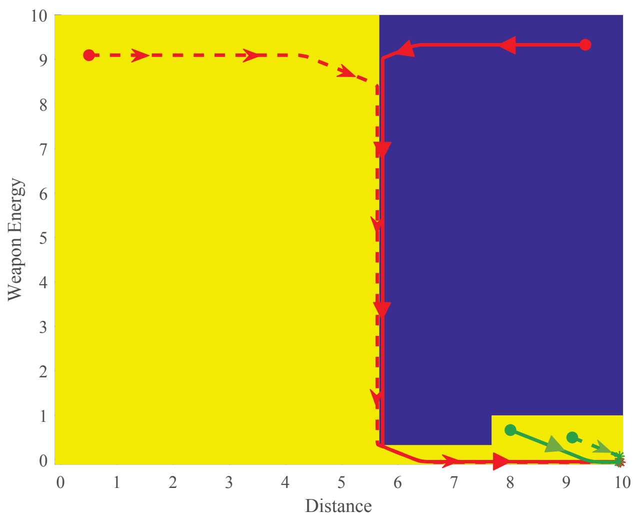
The solid green trajectory starts from the trained initial condition where the Attacker has only 4 weapon energy and starts at a distance of 8 from the Defender, $x_0 = \{8, 4\}$. The Attacker decides to retreat while using as much of its remaining weapon energy as possible. In this case, the damage that would be incurred trying to approach the Defender would be significantly greater than the amount of damage the Attacker could inflict with only 4 weapon energy remaining. The best strategy in this case is to take as small of a loss as possible and retreat while expending all remaining weapon energy. The integral utility for this trajectory can be seen in Figure 7 as the solid green line.

The dashed green trajectory shows a similar scenario, but in this case the Attacker starts in the untrained state of 1.7 weapon energy at a relative distance of 3.4, $x_0 = \{3.4, 1.7\}$. We can see that similarly to the trained initial condition, this trajectory implements a retreat strategy to minimize incurred cost. We can see that during its retreat, the Attacker passed through a blue *hold fire* grid cell in the fire control. In this case, the Attacker was already out of weapon energy, so the cell did not have an effect. However, if a trajectory with remaining weapon energy were to pass through this cell, a yellow *fire* cell would have reduced the incurred cost and ended up with a slightly higher integral utility.

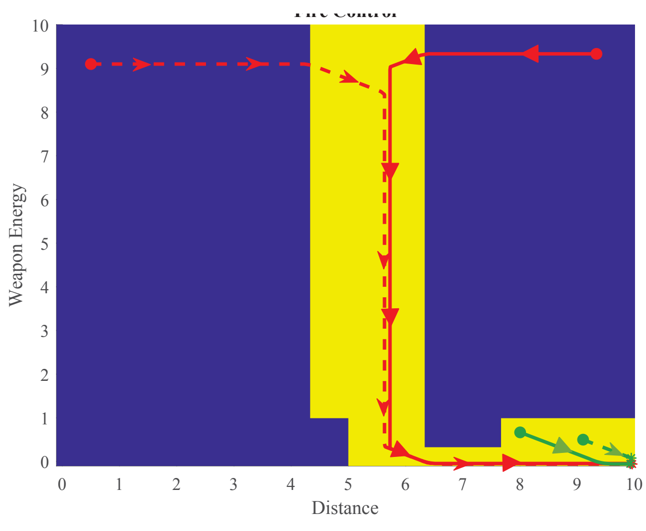
These small errors indicate that our controller was not completely done evolving and needed more time to refine the controller through crossing and mutation. While these errors may seem large, we can see that they are in locations that have very few trajectories passing through them. The fewer trajectories that pass through a cell, the less impact it has on the overall utility. The lower the effect a cell has on the utility, the less likely it is to be improved over another more impactful cell during the evolutionary algorithm. So overall, while these cells are indeed incorrect, they have a relatively small impact on the utility of trajectories generated by controller.

B. Ranged Attacker vs. Melee Defender

Next we examine the case in which the Attacker has a damage profile corresponding to a ranged unit. A ranged unit



(a) Motion Control



(b) Fire Control

Fig. 8: Best Evolved Ranged Attacker Grid Controllers

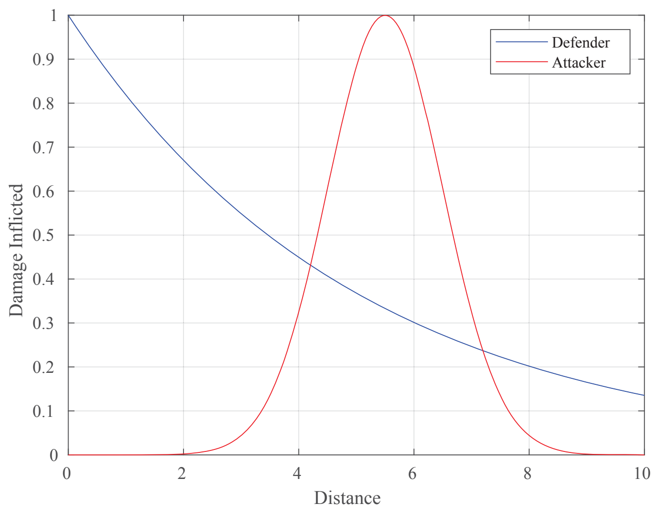


Fig. 9: Cost Function of Ranged Attacker vs Melee Defender

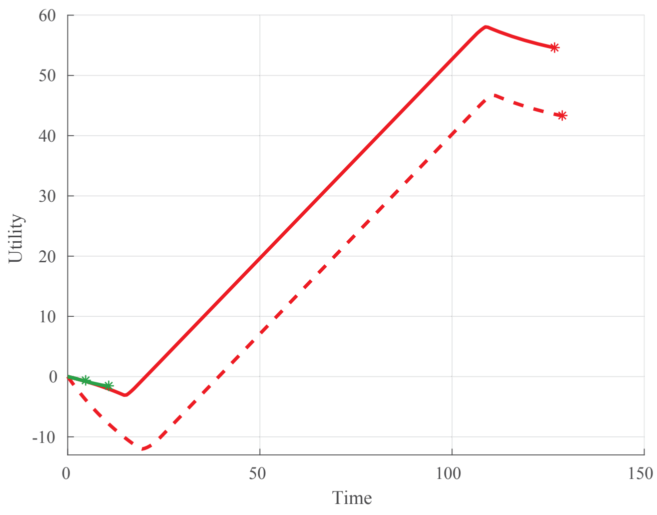


Fig. 10: Integral Utility

inflicts maximum damage from some non-zero distance that is usually outside the effective range of a melee unit. The damage profile chosen for our ranged unit is defined as

$$C_{A}(\mathbf{x}, \mathbf{u}) = u_2 \left(e^{-(d-5.5)^2/2} \right). \tag{11}$$

Figure 9 shows this ranged damage profile along side the standard melee damage profile from equation (9) for $u_2 = 1$. As shown, the ranged Attacker inflicts maximum damage at $d = 5.5$ and has a steep drop off in either direction.

The evolutionary algorithm was run using this ranged damage profile for the Attacker and the standard melee damage profile for the Defender. Similarly to the previous case, the maximum fitness over the evolution follows a logarithmic shape. The control matrices of the candidate controller with the highest overall fitness in the final generation are shown in Figure 8.

We begin the analysis with the solid red trajectory initialized from the trained initial condition of $\mathbf{x}_0 = \{9.66, 9.66\}$. The Attacker begins by holding its fire while approaching the Defender. At a distance of $d = 5.66$, the Attacker starts to fire. This strategy persists until it is almost out of weapon energy at which time it expends the rest of its weapon energy while running away to the retreat boundary.

Unlike the melee unit, this ranged control strategy did not hold the Attacker at its peak damage distance (in this case $d = 5.5$). Instead, it held it at a distance of $d = 5.66$. This behavior is a result of the controller parameterization. The only way for the parameterized grid controller to hold the Attacker at a fixed distance is to place it on a boundary between *move left* and *move right*. Ideally we would like to hold the Attacker at $d = 5.5$, but with the given grid resolution, the closest cell boundary occurs at $d = 5.66$. So the controller evolved a strategy in which it holds the attacker at the nearest possible distance of $d = 5.66$.

The dashed red trajectory shows the performance from the untrained initial condition of $\mathbf{x}_0 = \{0.5, 9.1\}$. Similarly to the trained case, the Attacker conserves its weapon energy until it approaches the maximum effective range of $d = 5.5$. Once it is near, it begins firing and then retreats when its weapon energy begins to deplete. This result again shows the ability of the evolved solution to translate its strategies into initial conditions it was never trained in.

The solid green line shows a retreat strategy initialized from the trained initial condition of $\mathbf{x}_0 = \{8, 0.66\}$. Much like the melee unit in section IV-A, there is not enough weapon energy left to justify an engagement. Instead the Attacker retreats while expending all remaining weapon energy to cut its losses.

The dashed green line shows a retreat strategy from the untrained initial condition of $\mathbf{x}_0 = \{9.1, 0.5\}$. Even though the controller was never trained at this initial condition, it is able to implement a retreat strategy. The integral utility of this and all previous ranged trajectories is shown by lines of corresponding color/line-style in Figure 10.

C. Summary of Results

The results in this paper show two distinct strategies that the evolutionary algorithm learned to implement. The first strategy resembles traditional kiting behavior, where the Attacker holds its fire while repositioning itself, expends its limited weapon energy where it can inflict maximum damage, and then runs away. The second strategy consists of retreating and is used to prevent the Attacker from engaging in scenarios where an attack would be too costly. As stated in the introduction, neither of these strategies were imposed upon the Attacker. Instead, they are a result of the cost functions, the utility function, and the evolutionary process. By looking at the parameterized control matrices, one can see the boundaries between these strategies and see how the evolutionary algorithm was able partition the state space.

It is important to note that these best evolved controllers are not guaranteed to be the optimal feedback controller $\mathbf{u}^*(\mathbf{x})$. Evolutionary optimization techniques such as this one suffer from the problem of induction and can only be guaranteed to improve the solution from initial conditions in which they are evaluated. Since we can not evaluate the evolved solution from the entire continuum of admissible states, we can not guarantee true optimality. Additionally, the controller parameterization limits the accuracy of the evolved controller. What our solution does provide is an estimate of the true underlying optimal control. Our results show that the evolved controllers are accurate enough to exhibit kiting behavior, and robust enough to extend this performance to untrained initial conditions. Thus, the evolved solutions can be effectively used as an approximation of the optimal control.

V. Conclusion

In this paper, we evolved closed loop controllers capable of controlling an attacking player with varying damage profiles. The resulting solutions were complex enough to implement multiple strategies and robust enough to be effective in both trained and untrained initial conditions. The grid parameterization used provided a simple way to parameterize a feedback controller so that an evolutionary algorithm could be used. Future work will aim to examine more complex controller parameterizations in hopes that we can better represent the true optimal boundaries and thus obtain more accurate solutions. We also plan to examine the case of a mobile Defender with more complex damage profiles.

References

[1] A. Uriarte and S. Ontanon, "Kiting in RTS Games Using Influence Maps," in *Artificial Intelligence in Adversarial Real-Time Games: Papers from the 2012 AIIDE Workshop*, 2012, pp. 31–36.

[2] T. DeWitt, S. J. Louis, and S. Liu, "Evolving micro for 3d real-time strategy games," in *2016 IEEE Conference on Computational Intelligence and Games (CIG)*, Sept 2016, pp. 1–8.

[3] J. Hagelback and S. J. Johansson, "Dealing with fog of war in a real time strategy game environment," in *2008 IEEE Symposium On Computational Intelligence and Games*, Dec 2008, pp. 55–62.

[4] J. Hagelback, "Potential-field based navigation in starcraft," in *2012 IEEE Conference on Computational Intelligence and Games (CIG)*, Sept 2012, pp. 388–393.

[5] E. Bryson and Y. Ho, *Applied Optimal Control*. New York: Hemisphere Publishing Corporation, 1975.

[6] D. Kirk, *Optimal Control Theory: An Introduction*, ser. Dover Books on Electrical Engineering Series. Dover Publications, 2004. [Online]. Available: https://books.google.com/books?id=fCh2SAtWIdwC

[7] Z. E. Fuchs and P. P. Khargonekar, "Encouraging attacker retreat through defender cooperation," in *50th Conference on Decision and Control and European Control Conference (CDC-ECC)*, Dec 2011, pp. 235–242.

[8] ——, "An engage or retreat differential game with an escort region," in *53rd Conference on Decision and Control*, 2014, pp. 4290–4297.

[9] Z. E. Fuchs, D. W. Casbeer, and E. Garcia, "Singular analysis of a multi-agent, turn-constrained, defensive game," in *American Control Conference (ACC), 2016*, July 2016.

[10] M. Pachter, E. Garcia, and D. W. Casbeer, "Active target defense differential game," in *52nd Annual Allerton Conference on Communication, Control, and Computing*, 2014, pp. 46–53.

[11] P. Androulakakis and Z. E. Fuchs, "Evolutionary design of engagement strategies for turn-constrained agents," in *IEEE Congress on Evolutionary Computation*, May 2017.

[12] P. Androulakakis, Z. E. Fuchs, and J. E. Shroyer, "Evolutionary design of an open-loop turn circle intercept controller," in *IEEE Congress on Evolutionary Computation*, July 2018.

[13] P. Androulakakis and Z. E. Fuchs, "The effects of controller representation on the evolution of the variable speed turn-circle intercept problem," in *IEEE Congress on Evolutionary Computation*, July 2019.

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