Comparing Two Representations for Evolving Micro in 3D RTS Games

Siming Liu, Sushil J. Louis
Evolutionary Computing Systems Lab (ECSL)
Department of Computer Science and Engineering
University of Nevada, Reno, NV, USA
{simingl, sushil}@cse.unr.edu

***Abstract*—We are interested in using genetic algorithms to generate winning maneuvering behaviors (or micro) in skirmish scenarios for three dimensional Real-Time Strategy games. In prior work, we encoded parameterized 3D micro behaviors like target selection and kiting into an algorithm for controlling friendly units in battle. Genetic algorithms then tuned these parameters to guide unit maneuvering in order to win skirmishes. In this study, we investigate a new representation for micro behaviors that uses only an influence map and a combination of thirteen potential fields. Genetic algorithms then tune influence map and potential field parameters to evolve winning micro behaviors. We compare the performance of both representations on identical scenarios against identical opponents in a full 3D RTS game environment called FastEcslent. The results show that the genetic algorithm using our new representation using less domain knowledge, reliably evolved high quality 3D micro behaviors that slightly, but significantly, outperformed behaviors from our prior work. Our work thus provides evidence for the viability of using potential fields for generating high quality, complex, micro for three dimensional RTS games.**

I. Introduction

The Real-Time Strategy (RTS) game genre has become a popular research platform for the study of Computational and Artificial Intelligence (CI and AI). In RTS games, players need to establish bases, collect resources, and train military units with the aim of eliminating their opponents. RTS games incorporate elements of long term planning and short term decision making that adds complexity to game play thus bringing many research challenges in the RTS game domain [1], [2]. First, the dynamic environment of RTS games requires real-time planning on several levels - strategic, tactical, and reactive. Second, players have to make decisions with incomplete information in the game world due to the existence of "fog of war". Third, players must adapt to their opponents' playing "style" in order to gain the advantage in later games. Fourth, players must employ spatial and temporal reasoning to exploit complex terrain and time-sensitive actions on a tactical and strategic level. All of the challenges described above and their impact on decision making are crucial for winning RTS games.

We usually divide the decision makings in RTS game into two categories, macro and micro. Macro refers to long term planning while micro focuses on short term operations. Good macro helps players to build a stronger army or develop good counter strategies to beat their opponents. On the other hand, good micro minimizes friendly unit loss and maximizes damage to opponents in a fight. Better micro can beat a numerically superior opponent and ensure decisive victories. This research focuses on using Genetic Algorithms (GAs) to generate high quality micro behaviors in skirmish scenarios for 3D RTS games.

In prior work, we used influence maps (IMs), potential fields (PFs), and reactive control scripts for spatial reasoning and unit maneuvering both in 2D and 3D RTS games [3], [4], [5]. We used 3D influence maps extended from the original 2D implementation to represent spatial information within a three-axis (x, y, z) cartesian coordinate space in [6]. Since other available RTS game research environments are 2D, we used a 3D open-source RTS game and simulation environment, FastEcslent, to enable work in a full 3D RTS game environment. FastEcslent not only provides us 3D game-physics, but also enables easy changes to physics, control, and graphics so that we can investigate the application of more realistic physics on the generation of 3D "micro" performance for real-world unmanned aerial vehicles. Figure 1 shows combat between two opposing teams in a 3D game world within FastEcslent.

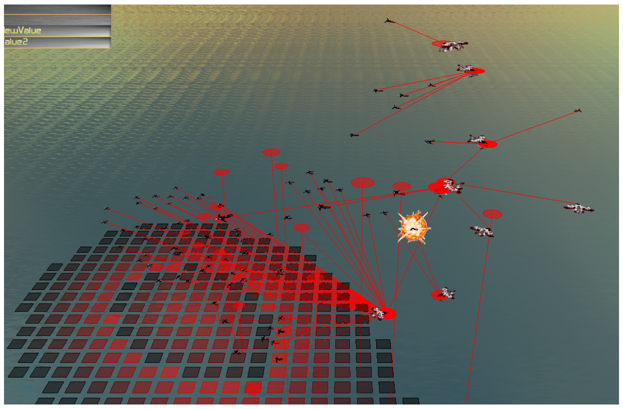
Fig. 1: Units are able to fly and fight in a full 3D environment within FastEcslent. Note that although the IM appears 2D, only the bottom layer of IM cells is shown to provide an unobstructed view of the skirmish.

Prior results show that GAs are able to evolve high quality micro behaviors in both 2D and 3D RTS games with our original parameterized algorithm approach. This meta-search approach which essentially searches a space of algorithms evolved micro for Vultures, a type of game unit with a ranged weapon. By finding the right parameter values, corresponding to the right micro algorithm, the GA enabled Vultures to learn three broad types of behaviors. First, Vultures learned to kite (hit and run) against a type of melee unit called a Zealot. This is very effective since the close-in melee units (Zealots) find it difficult to get in range to use their melee weapon against the longer ranged and faster Vultures. Second, Vultures learned to split enemy units into small groups to avoid being surrounded and to spread their own firepower. Finally, Vultures also evolved conservative behaviors that preserve hit-points and not get over aggressive.

However, the meta-search approach used in this earlier work depended on our hard coded algorithm which used much domain knowledge gleaned from human RTS game players. Kiting, target selection, and fleeing behaviors were hard coded and the GA simple tuned parameters to optimize the behavior to the speed, range, and health of the maneuvering unit. The search space of parameters was 2^{51}. This search space cannot be exhaustively searched but the genetic algorithm finds good solutions in reasonable time. However, possible behaviors are limited by our hard coded meta-algorithm. Moreover, the quality of the algorithm is also limited by the code-author's domain.

In this paper, we replace the rich domain knowledge parameterized algorithm with a combination of an influence map and potential fields to control friendly units' micro in battle. We call the AI player using this new micro representation the PPFBot[1]. The PPFBot adopts three types of attractive and repulsive forces represented by potential fields that depend on distance, weapon cooldown, and unit hit-points for both friendly and enemy units. The combined forces determine unit maneuvering during a skirmish. We then use GAs to evolve the PPFBot for high quality micro behaviors in a 3D skirmish scenario. However, the new micro representation requires our GAs to search in a much larger space (2^{226}) which is $2^{226}/2^{51} \approx 4.8 \times 10^{52}$ larger than the meta-search approach. Despite the size and without other domain knowledge, our results show that our PPFBot learned good target selection and kiting without such behaviors being explicityly encoded. Comparing PPFBot with MSBot (which uses our prior meta-search approach) on identical scenarios and against identical opponents, results show that our PPFBot slightly, but statistically significantly, outperformed MSBot. Since PPFBot does not incorporate any predefined algorithms, we believe the pure potential field approach holds more promise than the meta-search approach.

The remainder of this paper is organized as follows. Section II discusses related work in RTS AI research and common approaches to micro. The next section describes our 3D simulation platform, FastEcslent, and the scenario used in our experiments. Section IV introduces the pure potential field representation for PPFBot and the parallel GA used in our research. Section V presents preliminary results and compares the solutions produced by PPFBot and MSBot. Finally, the last section draws conclusions and discusses future work.

II. Related Work

A large amount of work has been done in applying various AI techniques towards RTS games. Very early work in our lab used influence maps for spatial reasoning to evolve a *LagoonCraft* RTS game player [7]. Sweetser *et al.* worked on an AI player designed with an influence map and cellular automata, where the influence map was used to model the game world and help the AI player in making decisions in their RTS game *EmerGEnt* [3]. Their AI player is flexible and able to respond to natural phenomena and user actions while chasing a target. Bergsma *et al.* proposed a game AI architecture for performing tactical AI by using influence maps for a turn based strategy game [4]. Preuss *et al.* introduced a flocking and influence map based path finding algorithm for group movement in the RTS game *Glest* [8], [9]. Su-Hyung *et al.* used evolutionary neural networks to evolve non-player characters' strategies based on the information provided by a layered influence map algorithm in the RTS game *Conqueror*. Uriarte *et al.* used influence maps for generating kiting behavior and used this for their StarCraft[2] bot *Nova* [10]. Avery *et al.* used a set of influence maps for guiding a group of friendly entities to maneuver and attack enemy entities based on opponent's position [11]. Their method generated an influence map for each entity in order to produce different unit movement in a game. This method however, does not scale well to large numbers of units due to its computationally intensive calculation in real time. Raboin *et al.* presented a heuristic search method for multi-agent pursuit-evasion games in a partially observable space [12]. In this paper, we use a unit influence map to represent spatial information and guide our units for winning skirmishes in RTS games.

Potential fields have also been found to be a useful technique in RTS AI research [13], [5]. Most of the work is related to unit navigation and collision avoidance [14]. The potential fields approach was first introduced by Ossama Khatib in 1986 for real time obstacle avoidance in mobile robots [15]. The technique was then widely applied to avoid obstacles and collisions especially in multiple unit scenarios with flocking [16], [17], [18]. Hagelbäck *et al.* presented a Multi-Agent Potential Field based bot architecture in the RTS game *ORTS* [19] and integrated potential fields into their AI player at both tactical and unit reaction control levels [20]. In this research, we use thirteen potential fields considering not only the distance but also unit status including hit-points and weapon cooldown from both friendly and enemy units to represent micro behaviors in 3D RTS games. More details of thirteen potential fields can be found in Section IV-B.

Other techniques are also used in micro AI research. Gunnerud *et al.* presented a hybrid system using case based reasoning and reinforcement learning for learning to select a target in given situations during a battle [21]. Wender *et al.* studied the application of reinforcement learning algorithms to learning micro during combat in RTS games [22]. The results showed that their reinforcement learning algorithms were able to learn selected tasks like "Fight", "Retreat", and "Idle" during combat. In this work, we introduce a new representation for 3D micro behaviors by using a combination of one influence map and thirteen potential fields without encoding any domain knowledge. We then apply genetic algorithms to search for high performance micro behaviors in a full 3D RTS game FastEcslent.

[1]Bot is a generic term for a control algorithm in the game AI community

[2]A popular RTS game and research platform

III. Simulation Environment

In the RTS AI research community, *StarCraft* is a popular research platform due to the emergence of the StarCraft: Brood War Application Programming Interface (BWAPI) framework and the *AIIDE*[3] and *CIG*[4] StarCraft AI tournaments [1]. BWAPI allows AI programs to play StarCraft games through code. However, StarCraft is a 2D RTS game and was designed for human players back in 1998. Since we are interested in micro behaviors in 3D RTS games, we are not able to conduct our experiments on the StarCraft platform as we cannot change the physics of motion within proprietary code. In addition, choosing an open-source 3D RTS game enables us to investigate the application of more realistic physics on evolved "micro" performance for real-world unmanned aerial vehicles. In this study, we run our experiments in a full open-source 3D RTS game environment, FastEclsent, that we developed for scientific research. FastEcslent was developed at the Evolutionary Computing Systems (ECSL) lab for evolutionary algorithms research in games, simulations, and other applications.

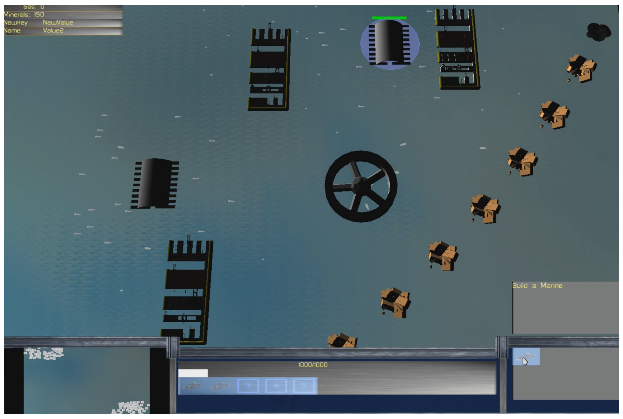

Fig. 2: A screenshot of a game play in FastEcslent.

FastEcslent is built on top of the popular open-source 3D graphics engine, OGRE [23] and supports both 2D and 3D game physics, enabling our research on evolving micro behaviors in both two and three dimensional coordinate spaces. It is open source and modular, so you can easily implement your own physics and AI control. We modeled the game play in FastEcslent to be similar to StarCraft to make comparisons easier. In FastEcslent, players are able to control several types of units to construct their bases and fight their opponents. FastEcslent can run without graphics (a seperate thread) thus providing easier integration with parallel genetic algorithms and other evolutionary computing algorithms. Figure 2 shows a screenshot of game play in FastEcslent.

Figure 1 shows a screenshot of a FastEcslent skirmish scenario with two opposing sides. In our scenario, each player controls a group of units initially spawned in different locations on a map with no obstacles. The entities used in FastEcslent reflect those in StarCraft, more specifically, Vultures and Zealots. A Vulture is a vulnerable unit with low hit-points but high movement speed, a ranged weapon, and considered effective when outmaneuvering slower melee units. A Zealot is a melee unit with short attack range and low movement speed but has high hit-points. Table I shows the details of these properties for both Vultures and Zealots which are used in our experiments. Since our research focuses on micro behaviors in skirmishes, we disabled "fog of war" in our scenario.

[3]http://www.StarCraftAICompetition.com
[4]http://cilab.sejong.ac.kr/sc_competition/

TABLE I: Unit properties defined in FastEcslent

Property	Vulture	Zealot
Hit-points	80	160
MaxSpeed	64	40
MaxDamage	20	16×2
Weapon's Range	256	224
Weapon's Cooldown	1.1	1.24

In order to evaluate the micro performance of PPFBot and MSBot during evolution we need a baseline opponent AI that controls the enemy Zealots to fight against our bots which control Vultures. We implemented a baseline AI named *Maintain AI* that behaves similar to the default StarCraft AI. A Zealot controlled by the Maintain AI selects the closest opponent unit as the current target to move toward and attack until the target is eliminated. We evolve our 3D micro behaviors for PPFBot and MSBot against a group of Zealots controlled by the Maintain AI in the given skirmish scenario. We limited the running time for our scenario to 6000 time steps (graphics frames) which is approximately 75 seconds at normal game speed.

IV. Methodology

With the availability of the 3D RTS research platform, the predefined skirmish scenario, and the baseline Maintain AI, we are able to create our bots to control a small group of Vultures to fight against a large group of melee units, Zealots, controlled by the Maintain AI for evaluating our 3D micro behaviors. In prior work, we used domain knowledge and encoded behaviors like target selection and kiting into an algorithm and used GAs to tune the micro script for each type of units to win skirmish scenarios. This study however, uses a different micro behavior representation which adopts little domain knowledge and combines only one influence map with a set of potential fields for group behavior representation.

A. Three Dimensional Influence Maps

We use a three dimensional unit influence map to generate a 3D spatial representation to tell our units where to move. The unit IM provides possible locations to move toward and the combined PFs control movement to locations provided by the IM. A typical IM is a grid defining the spatial information in a game world, with values assigned to each grid-cell by an IMFunction. An IMFunction is usually specified by two parameters, a weight, the value at the location of the entity and a range of influence in a 3D game world. The grid-cell value for each entity linearly decreases to zero as range increases. In this research, we extend our IMFunction from using two parameters (weight and range) to five parameters in order to represent more complicated spatial information as shown in Equation 1.

To calculate any grid-cell value V_c, we add the influence from each of the units within the range r from the cell. The influence of a unit is a weight calculated by the sum of the hit-points rate R_h scaled by $w1$, the weapon cooldown rate R_c scaled by $w2$, and a default weight $w3$. We then compute the influence of the grid-cell from the unit based on the distance d by removing $d \cdot \Delta f$ from the base weight of the unit. We can see that our IMFunction not only considers the units' positions in the game world but also includes the hit-points and weapon cooldown of each unit. For example, an enemy unit with low hit-points or its weapon in cooldown can generate a high influence map value. PPFBot always select the maximum IM value location as the target attack location, so our units (Vultures) will move towards and attack the damaged unit (low hitpoints) that cannot fire until the weapon cooldown period is over. In essence, changes in the five parameters ($w1$, $w2$, $w3$, r, Δf) of the IMFunction determines target selection for friendly units. "Attack maximum IM value location" as our target selection algorithm can be contrasted against the more complex hard coded logic in our MSBot's target selection algorithm.

$$V_c = \sum_{u \in \mathbb{R}} ((w1 \cdot R_h + w2 \cdot R_c + w3) - d \cdot \Delta f) \tag{1}$$

B. Three Dimensional Potential Fields

The unit IM provides possible locations to move toward and we use PFs to control unit movement to locations provided by the IM. Equation 2 shows a standard potential field function, where $\vec{F}$ is the potential force applied to the unit, with D being the distance from another unit. The direction of the 3D force is in the direction of the vector difference *from* the other unit. C and E are evolvable parameters.

$$\vec{F} = C \cdot D^E \tag{2}$$

Since our pure potential field approach encodes little domain knowledge, we want to incorporate as many factors as we can in order to enable PPFBot to evolve complex micro behaviors. Equation 3 shows our overall potential field ($\vec{PF}$) at a location composed from four different types of potential fields: PFs generated by distance ($\vec{PF}_d$), PFs generated by unit hit-point ($\vec{PF}_h$), PFs generated by unit weapon cooldown ($\vec{PF}_c$), and a PF generated by the current target ($\vec{PF}_t$). Note that both friendly and enemy units generate potential fields $\vec{PF}_d$, $\vec{PF}_h$, and $\vec{PF}_c$ and we have both attractive forces and repulsive forces. Therefore, each of $\vec{PF}_d$, $\vec{PF}_h$, and $\vec{PF}_c$ contains four PFs. Considering one attractive PF from $\vec{PF}_t$, the overall potential fields $\vec{PF}$ consists of thirteen potential fields in total. $\vec{PF}$ will then provide the desired heading and desired speed, which are used by the game physics in FastEcslent to control the unit's navigation. We apply GAs to evolve the two parameters C and E from Equation 2 for each potential force.

$$\vec{PF} = \vec{PF}_d + \vec{PF}_h + \vec{PF}_c + \vec{PF}_t \tag{3}$$

$\vec{PF}_d$ is typical potential fields generated by the distance from a unit to the PF location as shown in Equation 4. C_{ad}, C_{rd} are the coefficients and E_{ad}, E_{ad} are the exponents of attractive and repulsive force of $\vec{PF}_d$ accordingly. D represents the distance from an enemy unit's location to the our unit's location.

$$\vec{PF}_d = C_{ad} D^{E_{ad}} + C_{rd} D^{E_{rd}} \tag{4}$$

$\vec{PF}_h$ is potential fields generated by the health of a unit represented by the current hitpoints of that unit. When fired upon, the unit takes damage which reduces hitpoints and thus the health of the unit. This is shown in Equation 5. C_{ah}, C_{rh} are the coefficients and E_{ah}, E_{ah} are the exponents of attractive and repulsive force of $\vec{PF}_h$ accordingly. R_h is a fraction between 0 and 1 representing the health level of the unit calculated by $health_{cur}/health_{max}$.

$$\vec{PF}_h = C_{ah} R_h^{E_{ah}} + C_{rd} R_h^{E_{rh}} \tag{5}$$

$\vec{PF}_c$ is potential fields generated by the cooldown rate of a unit as shown in Equation 6. C_{ac}, C_{rc} are the coefficients and E_{ac}, E_{ac} are the exponents of attractive and repulsive force of $\vec{PF}_c$ accordingly. R_c is a fraction between 0 and 1 representing the cooldown level of the unit calculated by cd_{cur}/cd_{max}.

$$\vec{PF}_c = C_{ac} R_c^{E_{ac}} + C_{rc} R_c^{E_{rc}} \tag{6}$$

$\vec{PF}_t$ is a potential field generated by the current target location, as determined by the maximum IM value, shown in Equation 7. C_{ta} and E_{ta} are the coefficient and exponent of the attractive force of $\vec{PF}_t$ accordingly. D is the distance from the target location to our unit's location. Note that there is no repulsive potential force from the target location.

$$\vec{PF}_t = C_{ta} D^{E_{ta}} \tag{7}$$

All the potential field parameters used in the above equations are then encoded into a 226 bit-length binary string as a chromosome for our GAs. When FastEcslent receives a chromosome, it decodes the binary string into corresponding parameters accordingly and directs friendly units to move and attack enemy units in battle. The fitness of this chromosome is then computed and sent back to our GAs at the end of each match.

C. Fitness Evaluation

The goal of our predefined scenario is to maximize damage to enemy units while minimizing friendly unit damage received within a limited time duration. The fitness is calculated by the evaluation function shown in Equation 8 at the end of each game.

$$Fitness = TD_e + (HP_f \times 400) \tag{8}$$

where TD_e is the total damage dealt from friendly units to enemy units. HP_f represents the total remaining hit-points of all friendly units. According to our prior experiments, we scaled HP_f with a value 400 to give friendly unit hit-points more weight than enemy unit damage in order to encourage conservative moves in battle. The fitness function shown in Equation 8 is used by our parallel genetic algorithms to bias the evolutionary process for all experiments.

Note first, that we used this fairly arbitrary and experimentally determined 400 scaling factor in these and prior experiments to compute fitness. An alternative approach being explored in our current research is to apply multi-objective genetic algorithms and treat damage done and damage received as two criteria in a pareto-optimization setting.

Second, note that the same fitness can be generated in multiple ways. For example, a fitness of 8000 can be from 50 enemy units eliminated with no friendly units surviving or from 45 enemy units eliminated and three friendly units alive whose hit-points sum to 800.

D. Parallel Genetic Algorithm

Parallel Genetic Algorithms (PGAs) are extensions of canonical GAs and the well-known advantage of PGAs is their ability to speed up the evaluation process. We implemented our PGA as a single population master-slave PGA where there is only a single panmictic population on the master node. However, unlike the canonical GA, all individuals in the population are distributed to multiple slave nodes and evaluate in parallel. Since evaluation involves running the game engine to simulate a skirmish, this is the computationally expensive and time consuming part of the evolutionary search process. Parallel evaluation on n slave nodes usually results in linear speedup for genetic algorithms and we take advantage of this in our experiments. The evaluation of the population is distributed on a first come first served basis. Unevaluated individuals in a population are distributed to any unoccupied slave node from the master node. We use Open-MPI as our inter-processor communication backbone [24]. We also used CHC elitist selection in which offspring compete with their parents as well as each other for population slots in the next generation [25], [26]. CHC selection being strongly elitist preserves high fitness individuals from being eliminated during search. Early experiments showed that our CHC selection worked significantly better than the canonical roulette wheel selection on our study. In accordance with our prior experiments, we set the size of population to 50 and run the PGA for 60 generations. The probability of 4-point crossover was set to 0.88 and the probability of bit-mutation was 0.01.

V. Results and Discussion

FastEcslent is a deterministic RTS game environment which means that running a skirmish with a given set of parameters leads to identical fitness in every run. Unlike some other game engines, the FastEcslent game engine does not introduce any noise or randomness to the game such as probability of missing the target or the amount of damage by a shot. We ran our scenarios with 30 random seeds for both MSBot and PPFBot. Each skirmish lasts a *maximum* of 1.2 minutes on average although if one side is eliminated, skirmishes can be much shorter. With the population size of 50 run for 60 generations, we need upto $\frac{50\times60\times1.2}{60} = 60$ hours for each run of a canonical GA. Since we run our GA in parallel on eight cores with one master node and seven slave nodes, each run lasts on average 8.57 hours for 3000 evaluations.

A. Genetic Algorithm Results

The first set of experiments evaluated the performance of MSBot using the previous meta-search approach in our scenario as a baseline for comparison. Our skirmish scenario on the map described earlier has 8 friendly Vultures versus 100 enemy Zealots. These uneven numbers are necessary so that slow, short-ranged Zealots stand some chance against the longer ranged and faster Vultures. In line with our prior research, MSBot is able to evolve high fitness 3D micro behaviors within 3000 evaluations. The micro behavior of the MSBot tuned by our GAs for the ranged Vultures successfully destroyed a large number of enemy Zealots while avoiding damage. According to the fitness function, the theoretical maximum score for our scenario is 19200. This is obtained when eliminating all of the enemy Zealots (16000) with no friendly Vultures receiving damage (3200). Fig. 3 shows the average fitness from 30 PGA runs of MSBot.

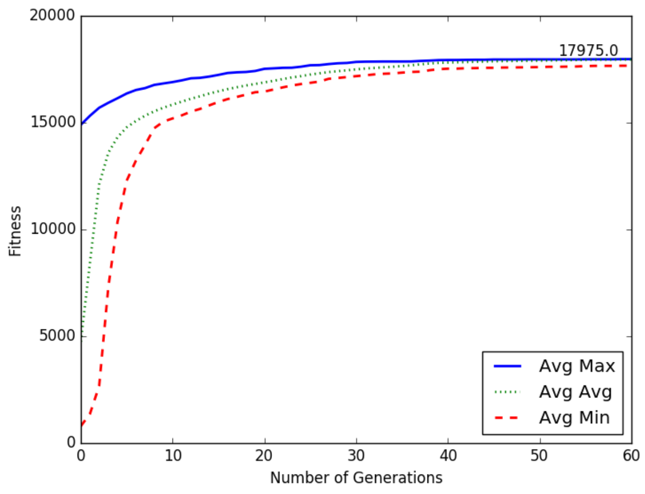

Fig. 3: Average performance of PGA on MSBot with 30 different random seeds. X-axis represents number of generations and Y-axis shows the average fitness over 30 runs.

Generation 0 in Figure 3 shows a large diversity of fitnesses over 30 runs from 792.5 to 14901.9, the difference between the average minumum and average maximum at the start. Since the initial population is generated randomly based on the given random seed, the large diversity of the first generation and the high initial maximum fitness shows that MSBot is able to perform fairly well simply by sampling 50 (population size) different parameter sets. That is, by randomly generating 50 sets of parameters and selecting the highest fitness individual may lead to a high fitness of 14901.9 corresponding to MSBot destroying maximum 93 Zealots out of 100 in the given time duration. The likely reason for the recurring high fitness in the first generation is that our MSBot encoded much domain knowledge in the micro control algorithm and there are many different parameter sets that result in high fitness. In such case, large population size of 50 used in this experiments greatly increased the possibility of generating high fitness solutions at random. As the evolutionary process goes on, fitness does increase and the very best solution found by the PGA is 18600 which is only 600 less than the theoretical maximum fitness of 19200. This indicates that the MSBot controlled 8 Vultures killed most of the Zealots with only a little damage received in the given time duration. The average of maximum fitness (the average best solution) over 30 runs of MSBot is 17975.0.

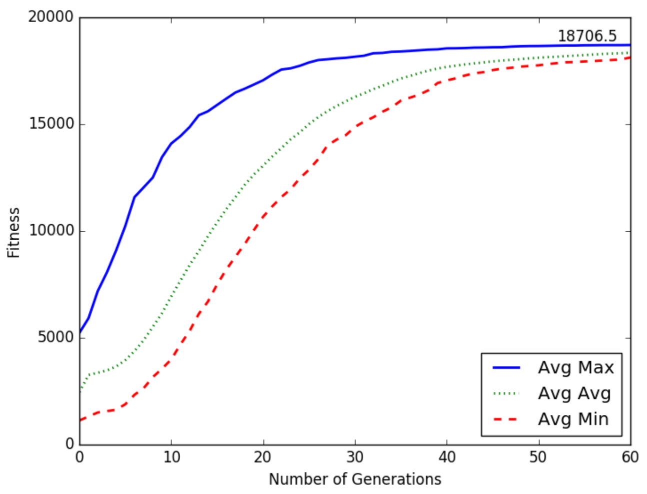

Fig. 4: Average performance of PGA on PPFBot with 30 different random seeds. X-axis is the number of generations and Y-axis shows the average fitnesses over 30 runs.

The average fitness curves of PPFBot shown in Figure 4 shows that the fitness of the initial population is between 1129.0 and 5227.1. We can see that the diversity of the fitness in the initial population of PPFBot is only one third compared to MSBot and the average maximum initial fitness is only 5227.1. Unlike with the MSBot, a small sample of 30 does not do well with the PPFBot. The fitness of 5227.1 means that the best solution in a randomly generated initial generation killed a maximum of 32 Zealots. However, the performance curves for PPFBot rise smoothly and the very best fitness over all 30 runs climbed up to 19120 which is only 80 less than the theoretical maximum fitness of 19200 and 720 higher than the best solution of MSBot (18600). The average over 30 runs of the maximum fitness in the PGA population (GA Max) is 18706.5 which is 731.5 higher than the average maximum fitness of MSBot. The higher fitness solutions show shows that even without domain knowledge about target selection, kiting, and fleeing explicitly encoded in the PPF representation, our search algorithms are able to evolve higher performance solution than the best solution of MSBot. Since MSBot heavily depends on rich domain knowledge, limitations on expert knowledge would also restrict the theoretical maximum performance of MSBot. On the other hand, PPFBot with fewer restriction imposed by encoded domain knowledge was able to find effective combinations of IM and PFs parameters leading to winning micro behaviors in our scenario.

Since both MSBot and PPFBot are able to find high performance solutions by the end of each run of the search algorithm, we are interested in the reliability of the two representations. Figure 5 shows the average of the maximum fitness over 30 runs of both MSBot and PPFBot with the standard deviation shown as error bars. We can see that our PGA reliably found a high fitness around 18000 on both MSBot and PPFBot. The standard deviation of MSBot's final fitnesses is 562.6, whereas the standard deviation of the PPF's final fitnesses is 647.3. Considering the average fitness to be as

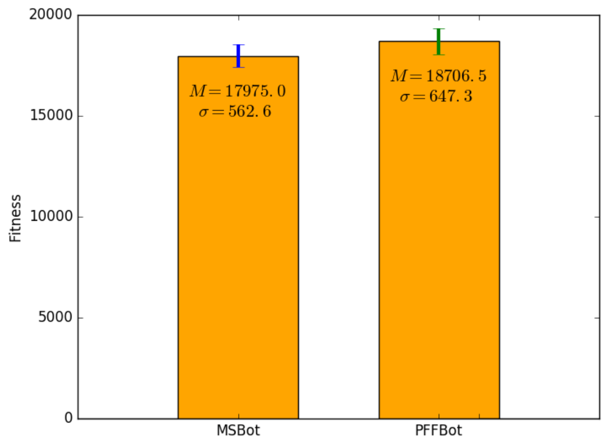

Fig. 5: PPFBot produced higher fitness than MSBot on average of 30 runs.

high as 18000, the low standard deviations of both MSBot and PPFBot provide evidence that the PGA reliably produces high quality 3D micro behaviors for both micro representations. The difference in performance between MSBot and PFFBot is small but statistically significant with a $P = 3.29 \times 10^{-7}$ using the t-test.

B. *Evolved 3D Micro Behavior*

We are also interested in the highest fitness 3D micro behaviors generated by both MSBot and PPFBot. The highest fitness found by MSBot is 18600. The behaviors created by this solution enabled friendly Vultures to spread across the map and split enemy units into smaller subgroups, thus decreasing the concentrated fire power of the more numerous enemy groups so that our units do not become overwhelmed. Figure 6 shows a screenshot of the skirmish that illustrates our Vulture's 3D micro behaviors evolved by MSBot. The best solution shows that our units were strongly attracted towards enemy units with small repulsion, leading friendly units to fight closely but remain out of the enemy unit's reach. A low freeze time (S_t) also encourages our units to move and kite more frequently to avoid being overwhelmed by the enemy units. A large value of the HP_{ef} parameter demonstrates that the evolved Vultures prefer to target closest enemy units instead of the damaged enemy units in our scenario, preventing potentially dangerous chases through enemy territory and kiting in quick intervals. Videos of evolved micro behaviors of MSBot can be found online at http://www.cse.unr.edu/~simingl/.

The best solution found by PPFBot is slightly better than MSBot. Without the limitation of the hard coded algorithm, PPFBot evolved different behaviors. All Vultures controlled by PPFBot move together to maximize their fire power. Therefore, 8 Vultures can kill 1 Zealot in a single round of shooting. Furthermore, without any hard coded kiting logic, PPFBot evolved a "hit and run" behavior which is essentially a "kiting" movement against Zealots. This indicates that with the general rule representation of influence maps and potential fields, kiting behavior which is effective for ranged units against melee

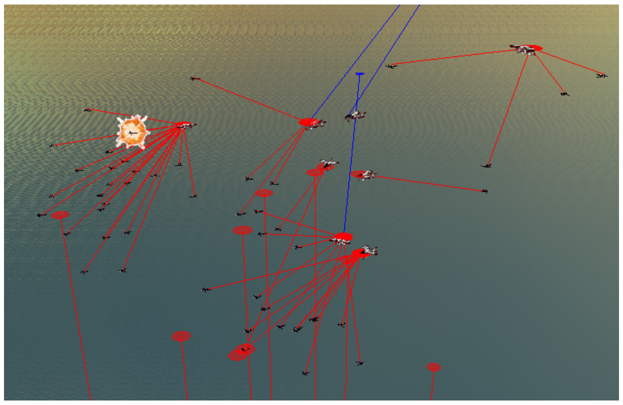

Fig. 6: The best solution evolved by MSBot with Vultures fight smaller fragments of the enemy group for a higher chance of survival.

attack units, emerges over time. With evolved PF parameter values, when a Vulture's weapon is in cooldown and can not fire, the Vulture is repelled away to keep a safe distance from enemy units. After the weapon becomes ready to fire, Vultures are attracted back towards Zealots to fight in battle. Figure 7 shows a screenshot of a fight that illustrates our Vulture's 3D micro behavior evolved by PPFBot. The videos at http://www.cse.unr.edu/~simingl/ also illustrate this emergent behavior and far better than a static screenshot.

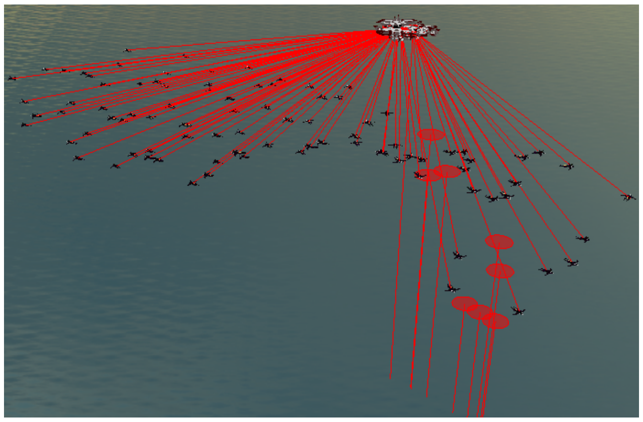

Fig. 7: The best solution evolved by PPFBot with Vultures stay together to concentrate fire powers.

C. Generalizability of Evolved 3D Micro Behaviors

We tested the generalizability of our evolved set of parameters for both MSBot and PPFBot by applying them to control the same amount of units in new scenarios with randomized initial unit positions. For each game, we randomly generate one location for each side in a 3D space with the limit of $x \subset (0, 1940)$, $y \subset (100, 1940)$, and $z \subset (0, 1940)$. The units of each side will spawn around the randomly generated location at the beginning of each game. We run 1000 games each with a different random seed leading to different initial unit locations. Figure 8 shows the average fitness and the standard deviations of 1000 runs for both MSBot and PPFBot. The average fitness over 1000 runs of MSBot is 15853.7 which is 80.02% of the highest fitness evolved in the original scenario (18600). The standard deviation of 1000 runs is 1449.1 which means MSBot generalizes fairly well with low variation when the initial position changes. However, the average fitness over 1000 runs of PPFBot is only 7902.0 which is 30.02% of the highest fitness evolved in the original scenario (19120). The standard deviation of 1000 runs is 3847.5 which means the best solution of PPFBot works well only on the training scenario and generalizes poorly to other scenarios with different initial positions.

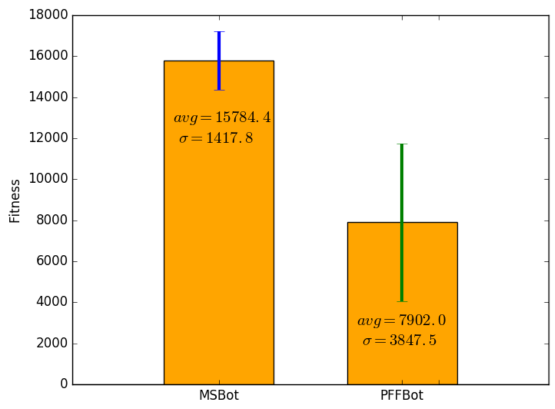

Fig. 8: The average fitness over 1000 runs on the best solutions of MSBot and PPFBot with random initial positions.

We hypothesize that the PPFBot overspecialized or prematurely converged to a sceanrio specific set of PF parameters and believe this lower generalizability can be mitigated by using multiple scenarios for each evaluation of the PPFBot. We are currently exploring this hypothesis by evolving the PPFBot using three scenarios per evaluation with encouraging results. In our case, on the one hand, it seems that too much domain knowledge (more contraints) can restrict solutions but can also ensure more generalizable results. On the other hand, less domain knowledge (fewer constraints) can lead to more exploration but may tend to find and converge on less generalizable results.

VI. Conclusion and Future Work

Our research focuses on generating effective micro behaviors including target selection, unit movement, and kiting in order to win skirmishes in 3D Real-Time Strategy games. We represented three dimensional micro behaviors as a combination of an IM and thirteen PFs in a 226 length bit-string without encoding any domain knowledge of 3D micro behaviors. Parallel genetic algorithms are applied to search for parameter values that lead to high quality micro on our scenario. These micro behaviors are then adopted by our PPFBot and compared with MSBot which uses the meta-search approach developed in our prior research. Since the chromosome length used by MSBot is only 51, the search space of PPFBot is $2^{175} \approx 4.8 \times 10^{52}$ times larger than the search space of MSBot.

We designed a scenario in which bots need to control 8 Vultures that fight against 100 Zealots to evaluate 3D micro performance in an open-source 3D RTS game, FastEcslent. The results show that parallel genetic algorithms quickly evolve high quality 3D micro for PPFBot although we encoded little domain knowledge in this representation. We ran the same experiments on MSBot with the meta-search approach developed in our prior research. Results show that our parallel genetic algorithms converge fast and evolve high quality 3D micro at the end of each run. The results also show that the fitness of the best solution evolved by PPFBot is slightly, but significantly, higher than MSBot. Considering that the PPFBot is representing 3D micro behaviors using only an influence map with thirteen potential fields, the higher fitness of PPFBot provides evidence that the pure potential field representation is a viable approach for evolving micro behaviors in full 3D RTS games. The Vultures evolved by MSBot exploit opposing Zealots by separating enemy units into smaller subgroups to avoid being surrounded and then kiting (hit and run behavior) till game time runs out. On the contrary, Vultures evolved by PPFBot stay together for concentrating their fire power to eliminate enemy Zealots while performing also kiting. Results also demonstrate that the best evolved solution of the MSBot generalizes well to new scenarios with different random unit initial positions. The best solution of PPFBot does not generalize as well.

We are interested in co-evolving 3D micro behaviors for both ranged units against melee units with our new pure potential field representation. We are also investigating replacing our simplified fitness function with a multi-objective formulation and using multi-objective evolutionary algorithms in a pareto-optimization setting. More techniques such as case-injected genetic algorithms or other knowledge-based systems can be applied to our approach to further investigate generalizability, speed, quality, and reliability of our micro representation and evolved solutions.

Acknowledgment

This material is based in part upon work supported by the National Science Foundation under Grant Numbers IIA-1301726 and RET-1542465. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation. This work is also partially supported by the Office of Naval Research under Grant N00014-15-1-2015.

References

[1] M. Buro, "Real-time strategy games: A new AI research challenge," *Proceedings of the 18th International Joint Conference on Artificial Intelligence. International Joint Conferences on Artificial Intelligence*, pp. 1534–1535, 2003.

[2] S. Ontañon, G. Synnaeve, A. Uriarte, F. Richoux, D. Churchill, M. Preuss *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. 1–19, 2013.

[3] P. Sweetser and J. Wiles, "Combining influence maps and cellular automata for reactive game agents," *Intelligent Data Engineering and Automated Learning-IDEAL 2005*, pp. 209–215, 2005.

[4] M. Bergsma and P. Spronck, "Adaptive spatial reasoning for turn-based strategy games," *Proceedings of AIIDE*, 2008.

[5] S. Liu, S. J. Louis, and M. Nicolescu, "Using CIGAR for finding effective group behaviors in RTS game," in *Computational Intelligence in Games (CIG), 2013 IEEE Conference on*. IEEE, 2013, pp. 1–8.

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

[7] C. Miles, J. Quiroz, R. Leigh, and S. Louis, "Co-evolving influence map tree based strategy game players," in *Computational Intelligence and Games, 2007. CIG 2007. IEEE Symposium on*, april 2007, pp. 88 –95.

[8] M. Preuss, N. Beume, H. Danielsiek, T. Hein, B. Naujoks, N. Piatkowski, R. Stuer, A. Thom, and S. Wessing, "Towards intelligent team composition and maneuvering in real-time strategy games," *Computational Intelligence and AI in Games, IEEE Transactions on*, vol. 2, no. 2, pp. 82–98, 2010.

[9] H. Danielsiek, R. Stuer, A. Thom, N. Beume, B. Naujoks, and M. Preuss, "Intelligent moving of groups in real-time strategy games," in *Computational Intelligence and Games, 2008. CIG'08. IEEE Symposium On*. IEEE, 2008, pp. 71–78.

[10] A. Uriarte and S. Ontañón, "Kiting in RTS games using influence maps," in *Eighth Artificial Intelligence and Interactive Digital Entertainment Conference*, 2012.

[11] P. Avery and S. Louis, "Coevolving influence maps for spatial team tactics in a RTS game," in *Proceedings of the 12th annual conference on Genetic and evolutionary computation*, ser. GECCO '10. New York, NY, USA: ACM, 2010, pp. 783–790. [Online]. Available: http://doi.acm.org/10.1145/1830483.1830621

[12] E. Raboin, U. Kuter, D. Nau, and S. Gupta, "Adversarial planning for multi-agent pursuit-evasion games in partially observable euclidean space," in *Eighth Artificial Intelligence and Interactive Digital Entertainment Conference*, 2012.

[13] S. Liu, S. J. Louis, and M. Nicolescu, "Comparing heuristic search methods for finding effective group behaviors in RTS game," in *Evolutionary Computation (CEC), 2013 IEEE Congress on*. IEEE, 2013, pp. 1371–1378.

[14] J. Borenstein and Y. Koren, "The vector field histogram-fast obstacle avoidance for mobile robots," *Robotics and Automation, IEEE Transactions on*, vol. 7, no. 3, pp. 278–288, 1991.

[15] O. Khatib, "Real-Time obstacle avoidance for manipulators and mobile robots," *The international journal of robotics research*, vol. 5, no. 1, pp. 90–98, 1986.

[16] R. Olfati-Saber, J. A. Fax, and R. M. Murray, "Consensus and cooperation in networked multi-agent systems," *Proceedings of the IEEE*, vol. 95, no. 1, pp. 215–233, 2007.

[17] M. Egerstedt and X. Hu, "Formation constrained multi-agent control," *Robotics and Automation, IEEE Transactions on*, vol. 17, no. 6, pp. 947–951, 2001.

[18] C. Reynolds, "Flocks, herds and schools: A distributed behavioral model," in *ACM SIGGRAPH Computer Graphics*, vol. 21, no. 4. ACM, 1987, pp. 25–34.

[19] M. Buro, "ORTS - A free software RTS game engine," *Accessed March*, vol. 20, p. 2007, 2007.

[20] J. Hagelbäck and S. J. Johansson, "The rise of potential fields in real time strategy bots," *Proceedings of Artificial Intelligence and Interactive Digital Entertainment (AIIDE)*, 2008.

[21] M. J. Gunnerud, "A cbr/rl system for learning micromanagement in real-time strategy games," 2009.

[22] S. Wender and I. Watson, "Applying reinforcement learning to small scale combat in the real-time strategy game StarCraft: Broodwar," in *Computational Intelligence and Games (CIG), 2012 IEEE Conference on*. IEEE, 2012, pp. 402–408.

[23] G. Junker, *Pro OGRE 3D programming*. Apress, 2006.

[24] R. L. Graham, G. M. Shipman, B. W. Barrett, R. H. Castain, G. Bosilca, and A. Lumsdaine, "Open mpi: A high-performance, heterogeneous mpi," in *Cluster Computing, 2006 IEEE International Conference on*. IEEE, 2006, pp. 1–9.

[25] J. H. Holland, "Adaptation in natural and artificial systems, university of michigan press," *Ann Arbor, MI*, vol. 1, no. 97, p. 5, 1975.

[26] L. J. Eshelman, "The CHC adaptive search algorithm : How to have safe search when engaging in nontraditional genetic recombination," *Foundations of Genetic Algorithms*, pp. 265–283, 1991. [Online]. Available: http://ci.nii.ac.jp/naid/10000024547/en/