Evolving Effective Micro Behaviors in RTS Game

Siming Liu, Sushil J. Louis, and Christopher Ballinger
Evolutionary Computing Systems Lab (ECSL)
Dept. of Computer Science and Engineering
University of Nevada, Reno
Reno NV, 89557
{simingl, sushil, caballinger}@cse.unr.edu

***Abstract*—We investigate using genetic algorithms to generate high quality micro management in combat scenarios for real-time strategy games. Macro and micro management are two key aspects of real-time strategy games. While good macro helps a player collect more resources and build more units, good micro helps a player win skirmishes against equal numbers and types of opponent units or win even when outnumbered. In this paper, we use influence maps and potential fields to generate micro management positioning and movement tactics. Micro behaviors are compactly encoded into fourteen parameters and we use genetic algorithms to search for effective micro management tactics for the given units. We tested the performance of our ECSLBot (the evolved player), obtained in this way against the default StarCraft AI, and two other state of the art bots, UAlbertaBot and Nova on several skirmish scenarios. The results show that the ECSLBot tuned by genetic algorithms outperforms the UAlbertaBot and Nova in kiting efficiency, target selection, and knowing when to flee to survive. We believe our approach is easy to extend to other types of units and can be easily adopted by other AI bots.**

I. Introduction

Real-Time Strategy (RTS) games have become a popular platform for computational and artificial intelligence (CI and AI) research in recent years. RTS Players need to gather resources, build structures, train military units, research technologies, conduct simulated warfare, and finally defeat their opponent. All of these factors and their impact on decision making are critical for a player to win a RTS game. In RTS communities, RTS players usually divide their decision making into two separate levels of tasks called *macro* and *micro* management as shown in Figure 1. Macro is long term planning, like strategies conducted in the early game, technology upgrading, and scouting. Good macro management helps a player to build a larger army or a better economy or both. On the other hand, micro management is the ability to control a group of units in combat or other skirmish scenarios to minimize unit loss and maximize damage to opponents. We decompose micro management into two parts: tactical and reactive control. Tactical control is concerned with the overall positioning and movement of a squad of units. Reactive control focuses on controlling a specific unit and moving, firing, fleeing during a battle. This paper focuses on using Genetic Algorithms (GAs) to find winning tactical and reactive control for combat scenarios. This focus is indicated by the dotted square in Figure 1. Micro management of units in combat aims to maximize damage given to enemy units and minimize damage to friendly units. Common micro techniques in combat include concentrating fire on a target, withdrawing seriously damaged units from the front of the battle, and kiting an enemy unit with shorter attack range. This paper focuses on applying GAs to generate competitive micro management as part of a RTS game player that can outperform an opponent with the same or greater number of enemy units. We plan to incorporate these results into the design of more complete RTS game players in our future work.

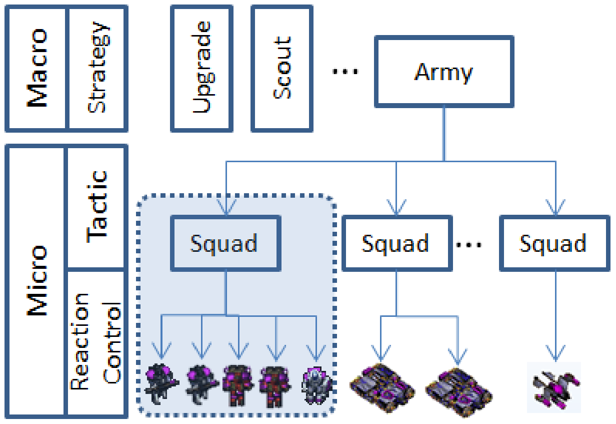


Fig. 1: Typical RTS AI levels of abstraction. Inspired by a figure from [1].

Spatial maneuvering being an important component of combat in RTS games, we applied a commonly used technique called Influence Maps (IMs) to represent terrain and enemy spatial information. IMs have been widely used to attack spatial problems in video games, robotics, and other fields. An IM is a grid placed over a virtual world with values assigned to each square by an IM function. Figure 2 shows an IM which represents a force of enemy units and the surrounding terrain in StarCraft: Brood War, our simulation environment. A unit IM is computed by all enemy units in the game. In our experiments, greater cell values indicate more enemy units in this area and more danger to friendly units. In addition to the position of enemy units, terrain is another critical factor for micro behaviors. For example, kiting enemy units near a wall is not a wise move. We then use another IM to represent terrain in the game world to assist micro management. We combined the two IMs and use this battlefield spatial information to guide our AI players positioning and reaction control. In this research, we use search algorithms to find optimal IM parameters that help specify high quality micro behaviors of our units for combat scenarios.

While good IMs can tell us where to go, good unit navigation can tell our units how best to move there. Potential

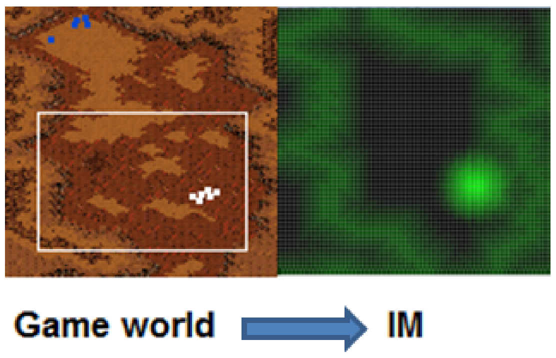


Fig. 2: IM representing the game world with enemy units and terrain. The light area on the bottom right represents enemy units. The light area surrounding the map represents a wall.

Fields (PFs) are used in our research to control a group of units navigating to particular locations on the map. PFs are used widely in robotics research for coordinating multiple entities movement and are often applied in video games for the same purpose. In our research, we apply two PFs to coordinate units' movement and use two parameters to specify each PF.

The goal of our research is to create a complete human-level RTS game player and this paper attacks one aspect of this problem: Finding effective micro management for winning small combat scenarios. Several challenges need to be resolved in this research. First, how do we represent reaction control like kiting, target selection, and fleeing? Furthermore, since the parameters for IM, PF, and reaction control are related, how do we tune these variables on different types of units and scenarios? To deal with these issues, we compactly represent micro behaviors as a combination of two IMs, two PFs, and a set of reaction control variables. We then use GAs to seek and find good combinations of these parameters that lead to winning micro behaviors. Since StarCraft: Brood War API (BWAPI) framework has become a popular platform for RTS AI research, we created three small battle scenarios in StarCraft and applied our GAs to search for optimal or near-optimal micro behaviors in these scenarios [2]. We then compared the performance of micro behaviors produced by our GAs with two state of the art StarCraft bots: *UAlbertaBot* [3] and *Nova* [4].

The remainder of this paper is organized as follows. Section II describes related work in RTS AI research and common techniques used in RTS micro research. The next section describes our simulation environment, design of our AI player, and our representation of micro for the GA. Section IV presents preliminary results and compares the solutions produced by our methods with two state of the art StarCraft bots. Finally, the last section draws conclusions and discusses future work.

II. Related Work

Much work has been done in applying different techniques to designing RTS AI players [5]. Let's first look at the work related to spatial reasoning and unit movement. Previous work has been done in our lab on applying IMs to evolve a LagoonCraft RTS game player [6]. Sweetser *et. al.* developed a game agent designed with IMs and cellular automata, where the IM was used to model the environment and help the agent in making decisions in their *EmerGEnt* game [7]. They built a flexible game agent that is able to respond to natural phenomena and user actions while pursuing a goal. Bergsma *et. al.* used IMs to generate adaptive AI for a turn based strategy game [8]. Su-Hyung *et. al.* proposed a strategy generation method using IMs in a strategy game *Conqueror*. He applied evolutionary neural networks to evolve non-player characters' strategies based on the information provided by layered IMs [9]. Avery *et. al.* worked on co-evolving team tactics using a set of influence maps, guiding a group of friendly units to move and attack enemy units based on opponent's position [10]. Their approach used one IM for each entity in the game to generate different unit movement in a game. This method however, does not scale well to large numbers of units. For example, if we have two hundred entities, the population cap for StarCraft, we will need two hundred IMs to be computed every update. This could be a heavy load for our system. Preuss *et. al.* used a flocking based and IM-based path finding algorithm to enhance group movement in the RTS game *Glest* [11], [12]. Raboin *et. al.* presented a heuristic search technique for multi-agent pursuit-evasion games in partially observable space [13]. In this paper, we use enemy units IM combined with terrain IM to gather spatial information and guide our units for winning micro management in RTS games.

Potential fields have also been applied to AI research in RTS games [14], [15]. Most of the work is related to unit movement for spatial navigation and collision avoidance [16]. This approach was first introduced by Ossama Khatib in 1986 while he was working on real time obstacle avoidance for mobile robots [17]. The technique was then widely used in avoiding obstacles and collisions especially in multiple unit scenarios with flocking [18], [19], [20]. Hagelbäck *et. al.* applied this technique to AI research within a RTS game [21]. They presented a Multi-Agent Potential Field based bot architecture in the RTS game *ORTS* [22] and incorporate PFs into their AI player at both tactical and unit reaction control level [23]. We use two PFs for group navigation in our work.

Reactive control, including individual unit movement and behaviors, aims at maximizing damage output to enemy units and minimizing the loss of friendly units. Common micro techniques in combat include fire concentration, target selection, fleeing, and kiting. Uriarte *et. al.* applied IMs for kiting, frequently used by human players, and incorporated the kiting behavior into their StarCraft bot *Nova* [4]. Gunnerud *et. al.* introduced a CBR/RL hybrid system for learning target selection in given situations during a battle [24]. Wender *et. al.* evaluated the suitability of reinforcement learning algorithms to perform the task of micro managing combat units in RTS games [25]. The results showed that their AI player was able to learn selected tasks like "Fight", "Retreat", and "Idle" during combat. However, the baseline of their evaluation is the default StarCraft AI and the tasks are limited. We scripted our reactive control behaviors with a list of unit features represented by six parameters. Each set of parameters influences reactive control behaviors including kiting, targeting, fleeing, and movement.

In this paper, we focus on coordinated group behaviors and effective reactive controls in a skirmish scenario and apply GAs to search for high performance micro behaviors against

TABLE I: Unit properties defined in StarCraft

Parameter	Vulture	Zealot	Purpose
Hit-points	80	160	Entity's health. Entity dies when Hit-points ≤ 0.
MaxSpeed	6.4	4.0	Maximum move speed of Entity.
Damage	20	8×2	Number of Hit-points that can be removed from the target's health by each hit.
Weapon	Ranged	Melee	The distance range within which an entity can fire upon target.
Cooldown	30	22	Time between weapons firing.
Destroy Score	150	200	Score gained by opponent when this unit has been killed.

different types of enemy units. IMs and PFs are used in our AI player to analyze the battlefield situation and generate group positioning. Reactive control behaviors are represented by six parameters and used by our micro agent to defeat enemy units.

III. Methodology

The first step of this research was building an infrastructure within which to run our AI player called a "bot". In our first set of scenarios, a bot controls a small group of units to fight against different numbers and types of enemy units. We list the basic rules of our scenarios below.

- No obstacles except a wall surrounding the map.
- Default StarCraft units.
- Perfect information of the map and the enemy units.

The primary objective is to defeat the opponent by eliminating their units while minimizing the loss of friendly units. The second objective is minimizing game duration. Our scenarios contains five *Vulture*s against different types of enemy units. A *Vulture* is a *Terran* unit with ranged attack weapon, low hit-points, and fast movement. Table I shows the detailed parameters for *Vulture* and another *Protoss* unit *Zealot* which is used in our experiments later.

A. Influence Maps and Potential Fields

We compactly represent micro behaviors as a combination of two IMs, two PFs, and a set of reactive control parameters. The IM generated from enemy units combined with the terrain IM tells our units where to go and PFs are used for unit navigation. The unit IM and the terrain IM are functions of the *weight*s and *range*s of enemy units and unwalkable terrain (walls). Since computation time depends on the number of IM cells, we use a cell size of 32×32 pixels.

A typical PF function is similar to equation 1, where F is the potential force on the unit, d is the distance from the source of the force to the unit. c is the coefficient and e is the exponent applied to distance and used to adjust the strength and direction of the vector force.

$$F = cd^e \tag{1}$$

We use two PFs of the form described by Equation 1 to control the movement of units. Each of the PF calculates one force acting on a unit. The two potential forces in our game world are:

- **Attractor:** The attraction force is generated by the destination where the unit is moving toward. It is inversely proportional to distance. A typical attractor looks like $F = \frac{2500}{d^{2.1}}$. Here $c = 2500$ and $e = -2.1$ with respect to Equation 1.
- **Repulsor:** This keeps friendly units moving to the destination from colliding with each other. It is usually stronger than the attractor force at short distances and weaker at long distances. A typical repulsor looks like $F = \frac{32000}{d^{3.2}}$.

Each PF is determined by two parameters, a coefficient c and an exponent e. Therefore, we use four parameters to determine a unit's PFs:

$$PF = c_a d^{e_a} + c_r d^{e_r} \tag{2}$$

where c_a and e_a are parameters of the attractor force, c_r and e_r for the friend repulsor force. These parameters are then encoded into a binary string for the GA.

B. Reactive Control

Besides the group positioning and unit movement, reactive control behaviors have to be represented in a way that our GAs can evolve. In our research, we considered frequently used reactive control behaviors: kiting, target selection, and fleeing which are usually used in real games by human players. Figure 3 shows the six variables used in our micro scripting logic. Table II explains the details and purposes of each variable.

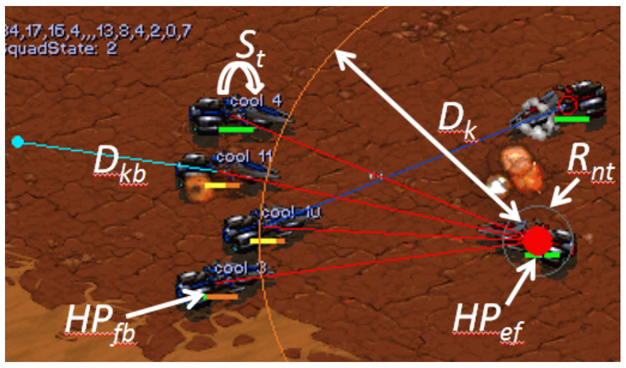


Fig. 3: Variables used to represent reactive control behaviors. The Vultures on the left side of map are friendly units. Two Vultures on the right are enemy units.

- **Kiting:** Also known as "hit and run". This behavior is especially useful in combat where the attack range of our units is larger than the attack range of enemy units. The variables used in kiting are S_t, D_k, D_{kb}.
- **Target Selection:** Concentrating fire on one target, or switching to a new target if there is an enemy with low hit-points nearby. The variables used in target selection are R_{nt}, HP_{ef}.

TABLE II: Chromosome

Variable	Bits	Description
W_U	5	Enemy unit weight in IMs.
R_U	4	Enemy unit influence range in IMs.
W_T	5	Terrain weight in IMs.
R_T	4	Terrain influence range in IMs.
c_a	6	Attractor coefficient.
c_f	6	Repulsor coefficient.
e_a	4	Attractor exponent.
e_f	4	Repulsor exponent.
S_t	4	The stand still time after each firing. Used for kiting.
D_k	5	The distance from the target that our unit start to kite.
R_{nt}	4	The radius around current target. Other enemy units within this range will be considered to be a new target.
D_{kb}	3	The distance for our unit to move backward during kiting.
HP_{ef}	3	The hit-points of nearby enemy units, under which target will be assigned.
HP_{fb}	3	The hit-points of our units, under which unit will flee.
Total	60	

- **Flee:** Fleeing from the front of the battle when our units have low hit-points. HP_{fb} was used to trigger this behavior.

Specifically, we encoded the chromosome into a 60-bit string. The detailed representation of IMs, PFs, and micro parameters are shown in Table II. Note that the sum IM is derived by summing the enemy unit IM and terrain IM so it does not need to be encoded. When the game engine receives a chromosome from our GA, it decodes the binary string into corresponding parameters according to the rule in Table II and directs friendly units to move to the position according to Algorithm 1 and then attack enemy units. Algorithm 1 shows the logic to locate an enemy target, an enemy unit with the lowest value in the IM before a battle. This algorithm also finds a weak spot to move toward before the fight through recursively calculating the lowest surrounding IM cell starting at the selected target until the cell value equals to 0. The fitness of this chromosome at the end of each match is then computed and sent back to our GA.

Algorithm 1 Targeting and Positioning Algorithm

```
Initialize TerrainIM, EnemyUnitIM, SumIM;
Target = MinIMValueUnit on SumIM;
movePos = Target.getPosition();
while (getIMValue(movePos) $>$ 0) do
    movePos = minSurroundingPos();
end while
moveTo(movePos);
attack(Target);
```

C. Fitness Evaluation

The objective of our first fitness evaluation is maximizing the damage to enemy units, minimizing the damage to friendly units, and minimizing the game duration in given scenarios. In this case, a unit remaining at the end of game will contribute 100 to its own side. The fitness of an individual will be determined by the difference in the number of units remaining on both sides at the end of each game. For example, suppose three friendly Vultures and one enemy Vulture remains at the end of the game, the score will be $(3-1)*100 = 200$ as shown in the first term of Equation 3. The detailed evaluation function to compute fitness (F) is:

$$F = (N_F - N_E) \times S_u + (1 - \frac{T}{MaxT}) \times S_t \quad (3)$$

where N_F represents how many friendly units remained, N_E is the number of enemy units remaining. S_u is the score for saving a unit as defined above. The second term of the evaluation function computes the impact of game time on score. T is the time spent on the whole game, the longer a game lasts, the lower is $1 - \frac{T}{MaxT}$. S_t in the function is the weight of time score which was set to 100 in the experiments. Maximum game time is 2500 frames, approximately one and a half minutes at normal game speed. We took game time into our evaluation because "timing" is an important factor in RTS games. Suppose combat lasts one minute. This might be enough time for the opponent to relocate backup troops from other places to support the ongoing skirmish thus increasing the chances of our player losing the battle. Therefore, combat duration becomes a crucial factor that we want to take into consideration in our evaluation function.

Minimizing the loss of friendly units may not be the primary objective in some scenarios. In some cases, we want to destroy as many enemy units as possible in a short time duration. For example, we want to test how many Zealots can be eliminated by 5 Vultures during 2500 frames. Killing one Zealot will add 200 to the score, while losing one Vultures will deduct only 150, therefore, the second fitness function is:

$$F = N_E \times DS_{ET} - N_F \times DS_{FT} \quad (4)$$

where N_F represents how many enemy units were killed, N_E is the number of friendly units being killed. DS_{ET} and DS_{FT} are the destroy scores for the types of unit being killed as show in Table I. We apply this fitness function on scenarios in which we want to evaluate how fast our bots can eliminate enemy units.

D. Genetic Algorithm

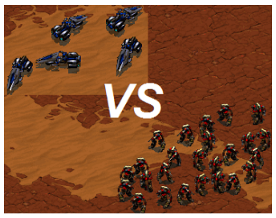
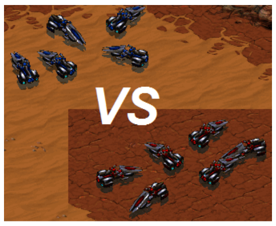
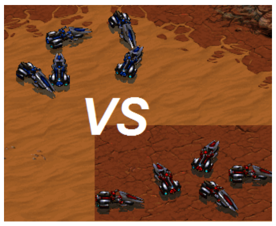
We used GAs to search for effective micro behaviors in combat scenarios against different types of enemy units. We used CHC elitist selection in which offspring compete with their parents as well as each other for population slots in the next generation [26], [27]. CHC selection being strongly elitist keeps high fitness individuals from being lost. Early experiments indicated that our CHC GAs worked significantly better than the canonical GAs on our problem. According to our previous experiments, we set the size of population to 20 and run the GA for 30 generations. The probability of crossover was 0.88 and the probability of bit-mutation was 0.01. Roulette wheel selection was used to select chromosomes for crossover.

E. StarCraft Bots

Thanks to recent StarCraft AI tournament competitions, several groups have been working on AI players for StarCraft.

In our research, we apply GAs to search for effective micro behaviors and compare the micro performance of our ECSLBot with two other state of the art bots: UAlbertaBot and Nova. UAlbertaBot was developed by D. Churchill from the University of Alberta and is the champion of the AIIDE 2013 StarCraft competition[1]. The micro logic of the UAlbertaBot is handled by MeleeManager and RangedManager for all types of units rather than each specific unit type. This abstraction makes it easy to adapt the micro managers to different types of military units. However, the UAlbertaBot implementation ignores the difference between units. For example, both Vulture and Dragoon are range attackers and can "kite" or "hit and run" against melee units, but they should kite differently based on weapon cool down time and target selection. Nova is another bot developed by A. Uriate. Nova was ranked number 7 on the AIIDE 2013 StarCraft competition. Nova uses IMs to control the navigation of multiple units and applied this idea to kiting behavior. The default StarCraft AI (SCAI) was used as one baseline in evaluating the performance of other bots. We encoded IMs, PFs, and reactive control behaviors to represent micro behaviors and apply GAs to search for optimal solutions against SCAI on different scenarios. We then compare the micro performance with UAlbertaBot and Nova on the same scenarios.

TABLE III: Snapshots of three scenarios.

Scenarios	Description	Bots
	5 Vultures versus 25 Zealots. Evaluating the efficiency of kiting behaviors.	UAlbertaBot Nova ECSLBot SCAI
	5 Vultures versus 6 Vultures. Evaluating the efficiency of target selection and hiding.	UAlbertaBot Nova ECSLBot SCAI
	5 Vultures versus 5 Vultures. Comparing the performance of each bot's micro behaviors by fighting against each other.	UAlbertaBot Nova ECSLBot

F. Scenarios

To evaluate the performance of ECSLBot's micro management from different perspectives, we designed three scenarios as shown in Table III. In the first scenario, GAs evolve high performance micro behaviors against melee attack enemy units. Kiting efficiency is extremely important in this type of battle. In the second scenario, GAs search for optimal solutions to fighting against ranged attack enemy units. Positioning and target selection become key contributors in these scenarios. The third scenario is created for our three bots to fight against each other in a fair environment. Instead of comparing bot performance against SCAI, the bots directly control equal numbers and types of units to fight each other and we compare the performance of each bot's micro.

[1]http://www.StarCraftAICompetition.com

IV. Results and Discussion

We used StarCraft's game engine to evaluate our evolving solutions. In order to increase the difficulty and unpredictability of the game play, the behavior of the game engine was set to non-deterministic for each game. In this case, some randomness is added by the game engine affecting the probability of hitting the target and the amount of damage done. This randomness is restricted to a small range so that results are not heavily affected. These non-deterministic settings are used in ladder games and professional tournaments as well. This does not impact some scenarios such as Vultures against Zealots too much, because theoretically Vultures can "kite" Zealots to death without losing even one hit-point. But the randomness may have amplified effect on other scenarios. For example, 5 Vultures fighting with 5 Vultures may end up with upto a 3 units difference at the end in fitness. To mitigate the influence of this non-determinism, individual fitness is computed from the average scores in 5 games. Furthermore, our GAs' results are collected from averaged scores over ten runs each with a different random seed. Early experiments showed that the speed of game play affects outcomes as well. Therefore, instead of using the fastest game speed possible: 0, we set our games to a slower speed of 10 to reduce the effect of the randomness [2]. Each game lasts 25 seconds on average, therefore, each evaluation will take $25 \times 5 = 125$ seconds to run. With the population size of 20 and run for 30 generations, we need $\frac{20 \times 30 \times 125}{60 \times 60} \approx 21$ hours for each run of our GA.

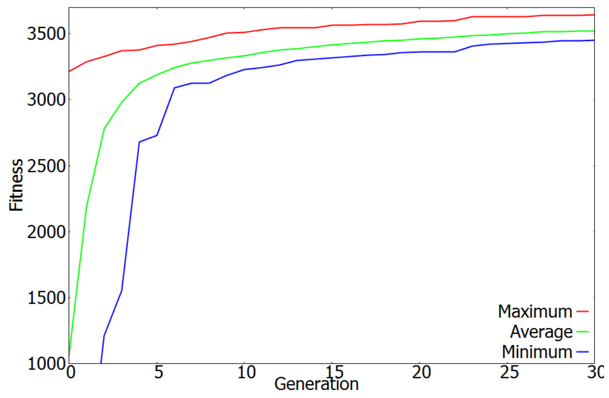


Fig. 4: Average score of bots versus generations on the 5 Vultures versus 25 Zealots scenario. X-axis represents time and Y-axis represents fitness.

[2]Human players play StarCraft at game speed 24.

TABLE IV: 5 Vultures vs 25 Zealots over 30 matches.

	Avg Score	Avg Killed	Avg Lost
UAlbertaBot vs SCAI	-83.33	3.33	5
Nova vs SCAI	3506.67	17.53	0.27
ECSLBot vs SCAI	3566.67	17.83	0.20

TABLE V: 5 Vultures vs 6 Vultures over 30 matches.

	Win	Draw	Lose	Killed	Remain
UAlbertaBot vs SCAI	0	0	30	2.67	0
Nova vs SCAI	1	0	29	3.13	0.13
ECSLBot vs SCAI	18	2	10	5.2	1.8

A. *Scenario 1: 5 Vultures vs 25 Zealots*

Kiting is one of the most frequent reactive control behaviors used by professional human players. We designed a kiting scenario in which a bot has to control 5 Vultures against 25 Zealots within 2500 frames[3]. Theoretically 5 Vultures cannot eliminate all 25 Zealots within 2500 frames because of their low damage output. Therefore, the purpose of this scenario is evaluating the efficiency of kiting behavior on destroying melee attack units. Equation 4 is used as our evaluation function in this scenario. Figure 4 shows the average scores of GAs running on this kiting scenario. We can see that the maximum fitness in the initial population is as high as 3217, which means our bot eliminated 16 Zealots within 2500 frames. However, the average of maximum fitness increases slowly to 3660 at generation 30, which is 18 Zealots. This results tell us that our GAs can easily find a kiting behavior to perform "hit and run" against melee attack units while trading off damage output. Our ECSLBot trades off well between kiting for safety and kiting for attack (damage).

We are interested in the performance differences among our ECSLBot (the best bot evolved by the GA) and the UAlbertaBot and Nova. We applied the UAlbertaBot and Nova to control the same number of Vultures (5) against 25 Zealots in the identical scenario. Table IV shows the results for all three bots versus the baseline SCAI over 30 runs. We can see that the UAlertaBot performed poorly against melee attack units in this scenario. This is mainly because UAlbertaBot uses the same micro logic for all its units. It eliminated only 3.33 Zealots on average in each game, while losing all of its Vultures. On the other hand, Nova's performance is surprisingly good. It killed 17.53 Zealots and lost only 0.27 Vultures on average in each game. This is because Nova hard coded and tuned micro logic specifically for Vulture and optimized Nova controlled Vulture kiting behavior against melee attack units. We then tested ECSLBot on this scenario. The results show that ECSLBot got the highest score on average over 30 runs. 17.83 Zealots being killed in one match on average, while losing only 0.20 Vultures. ECSLBot and Nova seem to have very similar kiting behavior and performance.

B. *Scenario 2: 5 Vultures vs 6 Vultures*

Besides performance against melee attack units, we are also interested in bot performance against ranged attack units. In this case, positioning and target selection become more important than kiting because the additional movement from kiting behavior will waste damage output while avoiding enemy's attack. We applied our GAs to search for effective micro behaviors using the same representation as in the previous scenario. However, we changed our fitness evaluation function to Equation 3 to maximize killing of enemy units, minimize the loss of friendly units, and minimize combat duration.

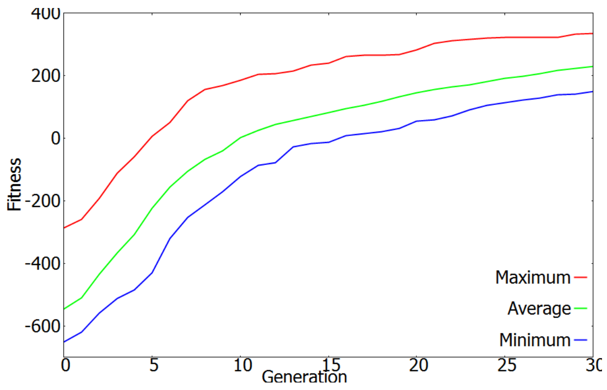


Fig. 5: Average score of ECSLBot over generations on the 5 Vultures versus 6 Vultures scenario. X-axis represents time and Y-axis represents fitness.

Figure 5 shows the average score of GAs in 5 Vultures versus 6 Vultures scenario. The average maximum fitness found by GAs is 336, which means 3 friendly units remained at the end of the game and all enemy units were eliminated. Considering that the Vulture is a vulnerable unit and easily dies, 3 Vultures saved after the battle is, we believe, high performance. Table V shows the results from all of our three bots tested in this scenario. All the bots run 30 times against the default SCAI. This time, both UAlbertaBot and Nova perform poorly. UAlbertaBot loses all 30 games against 6 Vultures, killing 2.67 enemy Vultures on average in each game, while losing all of its units. Nova performed slightly better than UAlbertaBot with 1 win and lost 29 out of 30 games. However, our ECSLBot evolved from GAs outperformed both of the other two bots with 18 wins, 2 draws, and 10 loses. 5.2 enemy Vultures are eliminated and 1.8 friendly Vultures survived on average in each match. This result indicates that in scenarios against ranged attack units, certain micro behaviors like kiting are not as effective versus melee attack units. Positioning and target selection become more important than kiting in such scenarios. UAlbertaBot and Nova did not optimize micro behaviors in all scenarios and performed poorly in these cases. Note that ECSLBot needs to adapt to new scenarios by evolving good values for the set of 14 parameters. However, this is simply a matter of running the algorithm for another 21 hours - this is low cost compared to AI programmer time.

C. *Scenario 3: 5 Vultures vs 5 Vultures*

We have compared the performance of three bots playing against SCAI on two different scenarios and the results show that ECSLBots outperformed both other bots using two different sets of parameters. However, what are the results when

[3]90 seconds with game speed 24.

TABLE VI: 5 Vultures vs 5 Vultures over 30 matches.

	Win	Draw	Lose	Units Remaining
UAlbertaBot vs Nova	24	5	1	2.33
ECSLBot vs Nova	30	0	0	3.37
ECSLBot vs UAlbertaBot	17	1	12	0.30

they play each other? To answer this question, we set up our third set of experiments on a 5 Vultures versus 5 Vultures scenario. Each bot plays against both of the other two bots 30 times with identical units. We applied the set of parameters evolved against 6 Vultures to play against UAlbertaBot and Nova. The result is that ECSLBot beats Nova but is defeated by UAlbertaBot. The replays show that the positioning of ECSLBot is too specific to static opponents controlled by SCAI and failed to beat UAlbertaBot. Therefore, we evolved another set of parameters directly against UAlbertaBot and applied ECSLBot with this set of parameters against UAlbertaBot and Nova. Table VI shows the detailed results among all the bots. We can see UAlbertaBot wins 24 matches, draws 5, and loses 1 against Nova. After examining game replays for these games, we found that Nova's micro kites against any type of opponent units. However, as our experiments with the second scenarios showed, kiting too much against the same ranged attack units actually decreased micro performance. UAlbertaBot on the other hand, disabled kiting when fighting against the equal weapon range units and defeated Nova easily. Similarly, ECSLBot defeated Nova on all 30 games without a loss or draw. Average units surviving was 3.37 which is higher than UAlbertaBot's 2.33. The final comparison was between ECSLBot versus UAlbertaBot. The results show that ECSLBot wins 17 matches, draws 1 match, and loses 12 matches out of 30. ECSLBot performed quite well on this scenario against the other bots.

D. Learned Micro Behaviors

We are interested in the differences in evolved parameters for the scenarios - against melee attack units and ranged attack units. Table VI lists the details of optimal solutions in different scenarios. Videos of all learned micro behaviors can be seen online [4]. There are two interesting findings in these results. The first concerns the learned optimal attack route in the scenario against 6 Vultures as shown in Figure 6. A gathering location at the left side of the map was learned by our ECSLBot to move toward before the battle. Our ECSLBot then controls 5 Vultures who follow this route to attack enemy units. The result is that only three of the enemy units were triggered in the fight against our 5 Vultures at the beginning of the fight. This group positioning helped ECSLBot minimize the damage taken from enemy units while maximizing damage output from outnumbered friendly units.

The second interesting finding is that different micro behaviors are learned by ECSLBot in different scenarios. Figure 7 shows that our ECSLBot kited heavily against Zealots as shown on the left side, but seldom move backward against ranged attack units as shown on the right side. The values of our parameters indicate the same thing. Table VII shows the parameter values found by our GAs in three scenarios. We can see that S_t (the first parameter in reactive control section) is 1 frames in scenarios against melee attack units, which means a Vulture starts to move backward right after every shot. On the other hand, S_t is much bigger (12 and 10 frames) against ranged attack units. This is because our units will gain more benefit after each weapon fire by standing still rather than moving backward immediately against ranged attack units.

[4]http://www.cse.unr.edu/∼simingl/publications.html

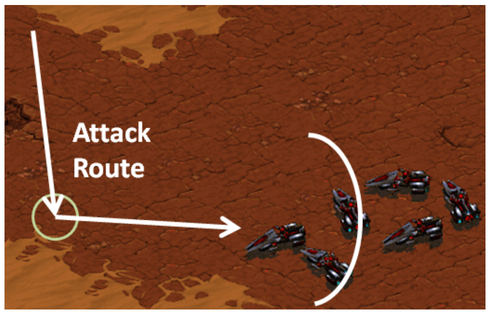


Fig. 6: Learned optimal attacking route against 6 Vultures.

TABLE VII: Parameter values of best evolved individuals.

Opponent	IMs				PFs				Reactive control					
25 Zealots	3	9	15	8	43	55	6	2	1	5	6	7	7	0
6 Vultures	16	13	20	10	50	26	13	4	12	9	1	7	6	7
5 Vultures	23	10	22	7	40	11	14	4	10	14	1	6	3	0

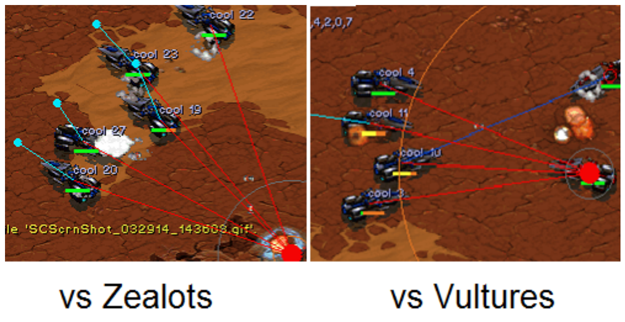


Fig. 7: Learned kiting behaviors against Zealots and Vultures. The left side of the figure shows that our Vultures are moving backward to kite enemy Zealots. The right side shows that our Vultures are facing the enemy Vultures with only one friendly Vulture moving backward to dodge.

V. Conclusion and Future Work

Our research focuses on generating effective micro management: group positioning, unit movement, kiting, target selection, and fleeing in order to win skirmishes in RTS games. We compactly represented micro behaviors as a combination of IMs, PFs, and reactive control parameters in a 60 length bit-string. GAs are applied to different scenarios to search for parameter values that lead to high performance micro. These micro behaviors are then adopted by our ECSLBot and compared with the default StarCraft AI and two state of the art StarCraft bots: UAlbertaBot and Nova.

We designed three scenarios in which bots need to control 5 Vultures against different types of enemies to evaluate micro performance. The results show that a genetic algorithm quickly evolves good micro for each scenario. With good scenario selection, we can then switch between parameter values according to opponent and scenario type and obtain good micro performance. Results show that Nova is highly effective at kiting against melee attack units but performs poorly against ranged attack units. UAlbertaBot, the AIIDE 2013 champion, performs poorly against melee attack units but is excellent against ranged attack units. Compared to the UAlbertaBot, we generate unit specific micro behaviors instead of a common logic for all units. With the right parameters, our ECSLBot beats both UAlbertaBot and Nova on all scenarios.

Our method used on the Terran unit Vulture can be quickly applied to other types of units. Unlike Nova, we do not hard code the micro behaviors for each individual unit type. What ECSLbot needs for developing new micro behaviors against a new type of unit is the values of another set of 14 parameters. Our GA can do this in about 21 hours. For example, our experiments with a ranged Protoss unit call the Dragoon instead of the Terran Vulture leads to similar results. We believe complete player bots using evolved ECSLBot micro parameters retrieved by an IM representing the battlefield will be harder to beat.

We are also interested in micro management with mixed unit types - which is still an open research question. In addition, we want to integrate the usage of unit abilities like the Terran Ghost's EMP pulse, into our micro behaviors. Furthermore, we want to be able to recognize and evolve terrain specific parameters to use terrain and sight to best advantage.

Acknowledgment

This material is based upon work supported by the National Science Foundation under grant number IIA-1301726.

References

[1] K. Rogers and A. Skabar, "A micromanagement task allocation system for real-time strategy games," *Computational Intelligence and AI in Games, IEEE Transactions on*, vol. 6, no. 1, pp. 67–77, March 2014.

[2] 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.

[3] D. Churchill and M. Buro, "Build order optimization in StarCraft," in *Artificial Intelligence and Interactive Digital Entertainment (AIIDE 2011)*, 10/2011 2011.

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

[5] 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.

[6] 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.

[7] 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.

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

[9] S.-H. Jang and S.-B. Cho, "Evolving neural NPCs with layered influence map in the real-time simulation game Conqueror," in *Computational Intelligence and Games, 2008. CIG '08. IEEE Symposium On*, dec. 2008, pp. 385 –388.

[10] 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

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] ——, "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.

[16] 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.

[17] 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.

[18] 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.

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

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

[21] J. Hagelbäck and S. J. Johansson, "Using multi-agent potential fields in real-time strategy games," in *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems - Volume 2*, ser. AAMAS '08. Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems, 2008, pp. 631–638. [Online]. Available: http://dl.acm.org/citation.cfm?id=1402298.1402312

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

[23] 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.

[24] M. J. Gunnerud, "A CBR/RL system for learning micromanagement in real-time strategy games," 2009.

[25] 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.

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

[27] 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/