Comparing Heuristic Search Methods for Finding Effective Group Behaviors in RTS Game

Siming Liu, Sushil J. Louis and Monica Nicolescu
Dept. of Computer Science and Engineering
University of Nevada, Reno
1664 N. Virginia Street, Reno NV 89557
{simingl, sushil, monica}@cse.unr.edu

***Abstract*—We compare genetic algorithms against hill-climbers for generating competitive unit micro-management for winning real-time strategy game skirmishes. Good group positioning and movement, which are part of unit micro-management can help win skirmishes against equal numbers and types of opponent units or even when outnumbered. In this paper, we use influence maps to generate group positioning and potential fields to guide unit movement. We tested the behaviors obtained from genetic algorithm and two types of hill-climbing search against the default Starcraft AI using the brood war API. Preliminary results show that while our hill-climbers quickly find influence maps and potential fields that generate quality positioning and movement in our simulations, they only find quality solutions fifty to seventy percent of the time. On the other hand, genetic algorithms evolve high quality solutions a hundred percent of the time, but take significantly longer.**

I. Introduction

Real-Time Strategy (RTS) games are a sub-genre of strategy computer and video games. They take place in real-time and involve resource gathering, micro-management and macro-management, tactics and strategies. Understanding each of these factors is critical for winning an RTS game. This paper focuses on building an RTS game player that can perform effective unit micro-management in a skirmish and compares genetic algorithms against two hill-climbers (HCs) for generating competitive solutions. A typical RTS game may include several skirmishes and losing one skirmish may eventually result in losing the entire game. Spatial positioning, group composition, and unit upgrades are some of the factors affecting skirmish outcomes. Spatial positioning decisions involve the spatial shape of the battleground: locating the weakness of enemy's defense, selecting attacking targets, and prepositioning your units for the upcoming fight. These difficult decisions make RTS games interesting and unpredictable. Micro-management of units in combat aims to maximize damage given to enemy units and minimizes damage to friendly units. Common micro techniques in combat include grouping units into formations, concentrating fire on one target, withdrawing seriously damaged units from combat, and using cheap units to draw the enemy's fire away from more expensive units.

Influence Maps (IMs) have been used for handling spatial problems in the Game Artificial Intelligence (Game AI) community [1]. They have been widely used in video games, robotics and other areas. For example, an IM model was built for playing Ms. Pac-Man [2]. Age of Empires uses IMs to find building locations based on nearby related resources [3]. An IM is a grid defining a specific piece of spatial information in a game world, with values assigned to each grid-cell by an IM function. Figure 1 shows an influence map which represents a force of enemy Marines in Starcraft: Brood War, our simulation environment. The grid-cell values are calculated by an IMFunction, parameterized for each type of game entity - in this case enemy Marines. IMFunction have two parameters, a `weight`, the value at the location of the entity and a `range` of influence. The grid-cell value for each entity linearly declines to zero as range increases. A grid-cell value can also be determined by more than one unit when IM ranges from different entities overlap. In this case, we sum the grid-cell values from all units influencing a particular cell. In our simulation environment, we set the weight of enemy units to be negative, therefore, lower cell values indicates more enemy Marines in the area and more danger to our units. We can use this enemy Marines position information to guide our AI player's spatial positioning. IMs have traditionally been hand-coded to solve particular problems. In this research, we use genetic algorithms and hill climbers to find near-optimal `weights` and `ranges` to help us find high quality group positioning of our units for skirmishes on a battlefield.

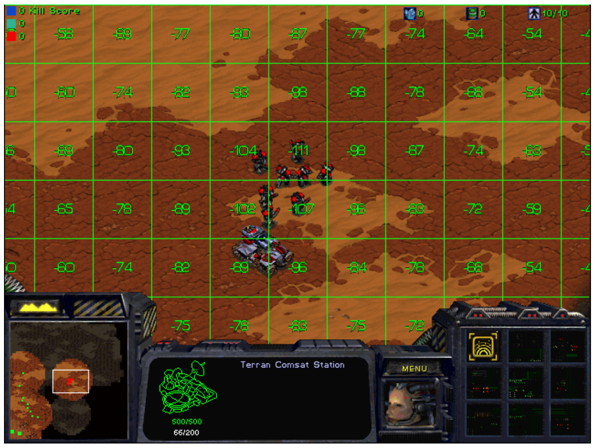


Fig. 1: Snapshot of an influence map represents enemy Marines.

While good unit positioning can put a player in a favorable situation in combat, good navigation can help the group move and attack more smoothly like a well-organized army. Potential Field (PF) is a technique from robotics research for coordinating multiple units' movement behaviors. A PF is a field defined in space that attracts or repulses entities in a game.

It specifies a vector force for every location in game space. A well-known example of potential field is gravitational potential. PFs have also been applied to RTS games mostly for spatial navigation and collision avoidance. We apply PFs to coordinate units' group movement in our work and use two parameters to specify each potential field for each unit type.

Our ultimate goal is to create human-level RTS game players and this paper compares GAs with two HCs for searching the space of IM and PF parameters to find good group positioning and movement to win a skirmish scenario. Several challenges have to be handled in such comparisons. First, how do we tune IM parameters for each type of unit to get good spatial information? Having more IM parameters or a smaller grid size could help us to get more information but will need more computational resources. Furthermore, potential field parameters and IM parameters are inter-dependent. To deal with these issues, we compactly represent group behaviors as a combination of three IMs and three PFs parameters and use a search algorithm to look for good combinations of these parameters that lead to winning group positioning and movement. Specifically, we compare the quality and robustness of solutions produced by genetic algorithms and two kinds of hill-climbers.

Another issue for the research community in RTS games is that most popular games like the Starcraft series or the Age of Empires series are not open source, therefore researchers cannot directly program their AI in the game. Things changed after the Starcraft: Brood War API (BWAPI) framework developed by a group of Starcraft community members was released [4]. We created three scenarios with BWAPI in Starcraft for this research.

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

II. Related Work

We first consider research in non-evolutionary computing approaches to design a competitive RTS game player. Aha et al. worked on a case-based plan selection method that learns to retrieve and adapt a suitable strategy for each specific situation during the game [5]. They built a case-base of previously encoded strategies, and the system learns to select the best matching strategy for different game states. They also performed an interesting analysis on the complexity of RTS games. Ontañón also worked on a real-time case based planning and execution techniques in RTS games and applied his technique in the WARGUS domain [6]. The system extracts behavioral knowledge from expert demonstrations in the form of individual cases and reuses the cases via a behavior generator. However, this system needs to record the actions of the expert player which is hard to do in closed source games. Furthermore, it requires that the expert explicitly tell the purpose of each action which made this method hard to expand. Miles and Louis used a case-injected GA to evolve a RTS player to take advantage of both case based reasoning and genetic algorithm [7]. An existing case base is not necessary in case-injected GA because the GA will search the space to find a near-optimal solution and save the optimal solutions into the case-base and case-injection can increased the learning rate of the GA. Avery and Louis worked on co-evolving team tactics using a combination of influence maps, guiding a group of units to move and attack based on opponent's position [8]. Their method used one influence map for each entity in the game which means that if we have two hundred entities, the population cap for Starcraft, we will need two hundred influence maps to be computed every frame. This could be a heavy load for a system. Preuss and Beume used a flocking based and influence map-based path finding algorithm to enhance team movement in the RTS game "Glest" [9], [10].

In contrast to research on development of complete RTS game players which considers every aspect of the game, we are only interested in spatial reasoning and movement related work. Previous work has been done in our lab to apply spatial reasoning techniques with influence maps to evolve a LagoonCraft RTS game player [11]. Sweetser and Wiles present a game agent designed with IMs, where the IM was used to model the environment and help the agent in making decisions [12]. They built a flexible game agent that is able to respond to natural phenomena while pursuing a goal. Bergsma and Spronck used influence maps to generate adaptive AI for a turn based strategy game [13]. Su-Hyung proposed a strategy generation method using influence maps in a strategy game, Conqueror. He used evolutionary neural networks to evolve non-player characters' strategies based on the information provided by layered influence maps [14].

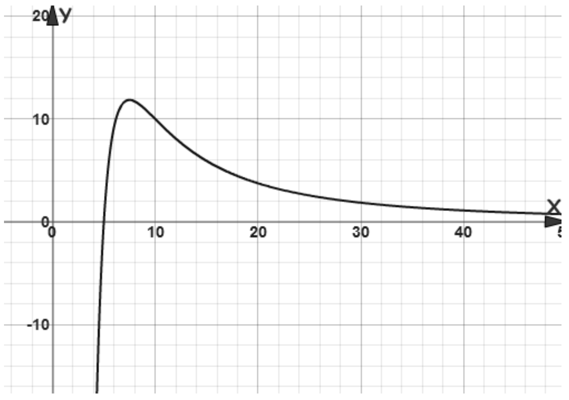


Fig. 2: Typical PF Function.

Potential fields have also been applied to AI agents in RTS games. Most of this work is related to spatial navigation and collision avoidance [15]. Figure 2 shows a typical potential function including both attraction and repulsion. The X-axis is distance between the destination and the entity, the Y-axis is the potential force. The negative part of the curve acts over a relatively short distance and represents repulsion. The positive part further away from the vertical axis represents the force of attraction. This approach was first introduced by Ossama Khatib in 1986 while he was looking for a real-time obstacle avoidance method for manipulators and mobile robots [16]. It was then widely used in avoiding obstacles and collisions especially for multiple units flocking [17], [18], [19]. Hagelback brought this technique into AI research within the

RTS game genre [20]. He presented a Multi-Agent Potential Field based bot architecture in ORTS [21]. It applied potential field at the tactical and unit operation level of the player AI [22].

In this paper, we focus on coordinated group behavior in a skirmish scenario and compare GAs with two HC search algorithms to find quality solutions. Coordinated group behavior includes spatial reasoning of group positions and good movement to those positions. Influence maps and potential fields are used in our AI player to analyze the battlefield situation, generate group positioning, and move units to win the battle against an equal opposing force controlled by the default Starcraft AI.

III. Methodology

The first step of this research was building an infrastructure in which to run our AI agent in an RTS game. We designed a simple two-player customized map with StarEdit which is a free tool provided by Blizzard Entertainment to build our own Starcraft map [23]. In our scenario, each player controls the same number of units starting at two different places on the map. We list the rules of our custom map below.

- The map does not have any obstacles and neutral creatures.
- The units are default Starcraft units.
- All the units use default Starcraft settings without upgrades.
- Units select targets based on Starcraft's built-in AI.
- There is no fog of war.

The goal is to eliminate opponent units while minimizing the loss of friendly units as well as minimizing the game duration. In case both sides have the same number and types of units left at the end of a game, we will get a higher score for a shorter game. Our scenario contains eight Marines and one Tank on each side. A Marine has low hit-points and a short attack range but can be built fast and cheaply. A Tank is stronger than a Marine, with high hit-points and attack range but it costs more to produce. Table I shows the detailed parameters for Marines and Tanks in Starcraft.

TABLE I: Parameters defined in Starcraft

Parameter	Marine	Tank	Purpose
Hit-points	40	150	Entity's unit of health. Hitpoints decrease when hit by opponent's weapon or when entity collides with another. Entity is destroyed when Hitpoints ≤ 0.
Size	12×20	32×32	Entity's size in pixel.
MaxSpeed	4.0	4.0	Maximum move speed of Entity.
MaxDamage	6	30	Maximum number of hitpoints that are removed from the target.
Range	128	224	The maximum distance at which an entity can fire upon another.
Cooldown	15	37	Time between weapons firing.
Destroy Score	100	700	Score gained by opponent when this unit been destroyed.

A. Influence Maps and Potential Fields

We compactly represent group behavior as a combination of three IMs and three PFs. Since each unit type has differing properties, we use one IM per unit type. Enemy unit generated IMs tell our units where to go and our units navigate to the indicated positions using PFs for movement. For the three preliminary scenarios considered in this paper, we only used three IMs for enemy units. Specifically, an enemy Marine IM, an enemy Tank IM, and an IM that sums the enemy Marine and Tank IMs. The enemy Marine and Tank IMs were specified by the `weights` and `ranges` of enemy Marine and Tank. Since computation time depends on the number of IM cells, we used a cell size of 64×64 pixels on the Starcraft map.

A typical PF function is similar to equation 1, where F is the potential force to the entity, d is the distance from the source of the force to the entity. c is the coefficient and e is the exponent effector.

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

We use three PFs of the form described by equation 1 to control the movement of entities in game. Each of these PFs describes one type of force acting on a unit. The three potential forces in the game world are:

- **Attractor:** The attraction force is inversely proportional to distance squared. A typical attractor looks like $F = \frac{2000}{d^2}$. Here 2000 and 2 are the PF's parameters c and e.
- **Friend Repulsor:** This keeps friendly units from colliding with each other. It is typically stronger than the attractor at short distances and weaker at long distances. A typical repulser looks like $F = \frac{30000}{d^3}$.
- **Enemy Repulsor:** This repels friendly units from enemy units. It is similar to the friend replusor.

Since each PF is determined by two parameters, a coefficient and exponent of d, six parameters determine a unit's potential field:

$$PF = \{C_{Att}, C_{RepF}, C_{RepE}, M_{Att}, M_{RepF}, M_{RepE}\} \tag{2}$$

where C_{Att} and M_{Att} are parameters of the attractor potential function, C_{RepF} and M_{RepF} for the friend replulsor, and C_{RepE} and M_{RepE} for enemy replusor. These parameters are then encoded into a binary string which worked with both GAs and HCs, and is the chromosome of our GAs. We encoded the chromosome in a 48-bit string. The detailed representation of IMs and PFs parameters are shown in table II. When BWAPI receives a chromosome, it decodes the binary string to corresponding parameters and directs friendly units to move and attack enemies. The fitness of this chromosome at the end of each match was then sent back to our search algorithm.

B. Fitness Evaluation

All the evaluation scores used in our fitness evaluation function are the default built-in score from Starcraft. The score

TABLE II: Chromosome

Parameter	Bits	Description
W_{Marine}	5	Weight of Marine in IMs
R_{Marine}	4	Range of Marine in IMs
W_{Tank}	5	Weight of Tank in IMs
R_{Tank}	4	Range of Tank in IMs
C_{Att}	6	Coefficient of attractor field
C_{RepF}	6	Coefficient of friendly repulser field
C_{RepE}	6	Coefficient of enemy repulser field
M_{Att}	4	Exponential of attractor field
M_{RepF}	4	Exponential of friendly repulser field
M_{RepE}	4	Exponential of enemy repulser field
Total	48	

for destroying a unit is based on how many resources are used. For example, one Marine needs 50 minerals to be produced. Destroying an enemy marine contributes 100 to the fitness function. One Tank needs 150 minerals and 100 gas. Gas counts double compared to minerals, therefore the score for destroying a Tank is $150 \times 2 + 100 \times 4 = 700$. The detailed evaluation function to compute fitness (F) is:

$$\begin{aligned} F &= (N_{FM} - N_{EM}) \times S_M + (N_{FT} - N_{ET}) \times S_T \\ &\quad + (1 - \tfrac{T}{MaxFrame}) \times S_{time} \end{aligned} \tag{3}$$

where fitness is calculated at the end of the each skirmish (game). N_{FM} represents how many enemy Marines were killed by the friendly side, N_{EM} is the number of friendly Marine killed by the enemy. S_M is the score for destroying a Marine as defined above. N_{FT}, N_{ET} and S_T have the same meaning for Tanks. The third part 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 $1 - \frac{T}{MaxFrame}$. S_{time} 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 a normal game speed. Therefore, time score being 0 means the game used up $T = 2500$ frames, 100 represents game ended within one frame. The reason to take game time into evaluation is because "timing" is an important factor in RTS game. Suppose a battle lasts as long as one minute, there is enough time for the opponent to build more units or relocate troops from other places to support this battle thus increasing the chances of the player losing the battle. Therefore, battle duration becomes a crucial effect that we want to take it into consideration in our evaluation function.

However, if the hill climber's starting point falls into a position where friendly units never engage the enemy units, F will be 0 and the hill climber will never find a close solution that it can use to climb out of this local minimum. Therefore, we need the fitness evaluation to also account for scenarios without engagement. Distance turned out to be a good indicator for evaluating F in such cases. The smaller the average distance between our units to enemy units the better, since this indicates that the group moves in the right direction and should therefore get relatively higher score. We used equation 4 to evaluate matches when there is no engagement during the game.

$$F = (1 - \frac{\overline{D}}{D_{max}}) \times S_{dist} \tag{4}$$

where $\overline{D}$ is average distance from friendly units to enemy units. $1 - \frac{\overline{D}}{D_{max}}$ converts maximize average distance to distance minimization. D_{max} is a constant value given the maximum distance in the map. S_{dist} is the weight of distance score which was set to 100 in the experiments.

C. Bit Setting Hill-climber

We use the Bit-Setting Optimization (BSO) hill climber to search a locally optimal solution by sequentially flipping each bit and keeping the better fitness solution [24]. Algorithm 1 shows the pseudo code of our BSO hill climber.

Algorithm 1 Bit Setting Optimization Hill-climber

```
$chromosome$ = init()
select first bit
eval($chromosome$)
**while** evaluation time $\leq$ Max **do**
    **while** not end of $chromosome$ **do**
        flip current bit
        eval($chromosome$)
        **if** fitness decreased **then**
            flip current bit back
        **end if**
        select next bit
    **end while**
**end while**
```

BSO is defined over a Hamming space where points in the space are represented by binary strings. The performance of BSO depends on the initial random seed, and it searches a subset of the search space based on this initial point. We initialize our BSO with ten different random seeds to randomly generate initial points. Using these ten different seeds enables us to obtain and report on statistically significant results. To make results from HCs and GAs comparable, we set the maximum number of evaluations of all these algorithms to 4000 and restart BSO from the first bit until we reach 4000 evaluations.

Algorithm 2 Random Flip Optimization Hill-climber

```
$chromosome$ = init()
eval($chromosome$)
**while** evaluation time $\leq$ Max **do**
    select random position
    flip the selected position
    eval($chromosome$)
    $fitness_{new}$ = eval($chromosome_{new}$)
    **if** fitness decreased **then**
        flip the selected bit back
    **end if**
**end while**
```

D. Random Flip Hill-climber

The bit setting hill-climber searches only a relatively small subset of the whole search space to find local optima, and it heavily depends on the initial starting point. However, Random Flip Optimization (RFO), a different hill-climber could search a different and larger space from the same initial points.

Algorithm 2 shows the pseudo code for our random flip hill-climber which starts from the same set of ten initial points as our BSO.

E. Genetic Algorithm

We used a CHC based GA in our experiments instead of canonical GA [25], [26]. The difference between CHC and canonical GA is CHC stands for cross generational elitist selection, heterogeneous recombination and cataclysmic mutation. CHC selects the N best individuals from the combined parent and offspring populations to create the next generation after recombination. Early experiments indicated that our CHC GA worked significantly better that the canonical GA on our problem.

Algorithm 3 CHC Genetic Algorithm

```
initial $population$
eval($population$)
**while** (current $\leq$ maxGeneration) **do**
  **if** generate $offspring$ **then**
    selection($population$)
    crossover($population$)
    mutation($population$)
  **end if**
  eval($offspring$)
  $tmpPopulation$ = rank ($population$, $offspring$)
  $offspring$ = top half of $tmpPopulation$
**end while**
```

According to previous experiments in our lab, we set the population size to 80 and ran the GA for 60 generations. The probability of crossover was 88% and we used CHC selection. We also used bit-mutation with 1% chance of each individual bit flipping in value. Standard roulette wheel selection was used to select chromosomes for crossover. CHC being strongly elitist keeps valuable information from being lost if our GA produces low fitness children. These operator choices and GA parameter values were empirically determined to work well.

IV. Results and Discussion

We used Starcraft's built-in AI to test our evolving solutions. However, the behavior of the default AI was set to non-deterministic for increasing interest and unpredictability. The Starcraft game engine added minor randomness in choosing the target, in the probability of hitting the target, and in the amount of damage done. For example, two Marines fighting each other might end up with different results in two games. But the randomness is restricted to a small range so that results are not heavily affected. This however means that in our case, evolved parameters will not guarantee the same high score every time we run. In other words, a high fitness solution has a high probability to get a high score in the game.

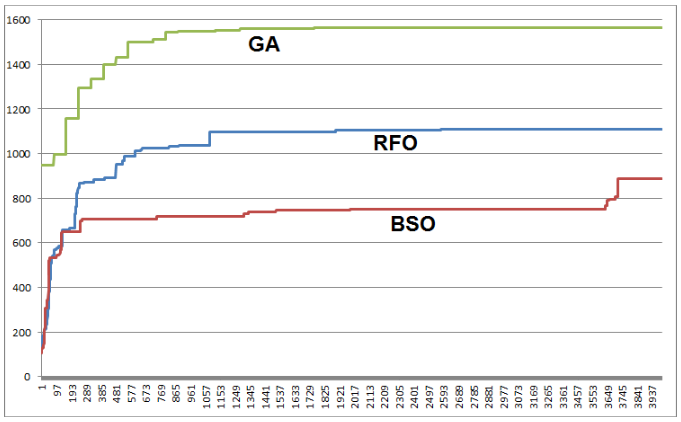


Fig. 3: Average score of BSO, RLO hill climbers and GA over time. X-axis represents the evaluation times and Y-axis represents the average fitness evaluated by fitness function.

According to the evaluation function and customized game map, the theoretic maximum score for destroying enemy forces (detroy score) is 1500 and maximum time score (corresponding to minimal time) is 100, therefore, the maximum of evaluation score or fitness is 1600. Note that the first two digits in a evaluation score represent the destroy score, and the last two digits represent time score. For example, if the final score ends up at 1451 we can infer the following

- The score being positive means our AI player defeated the built-in AI.
- 1400 represents destroy score and compared to maximum 1500, our AI player lost 100 which indicates that one Marine was killed by the enemy during the game.
- The last two digits being 51 represents $(1 - \frac{51}{100}) \times 2500 = 1225$ frames spent during the entire game and approximately 43.75 seconds converted to normal game speed.

In our experiments, we test each algorithm for ten different random seeds. Each such test lasted six hours to run the 4000 evaluations. We can see from the bar-graph in Figure 4 that the BSO HC could find good solutions 5 out of 10 times. This probably means that there are many local hills of not very high quality. The average score of BSO shown in figure 3 climbed fast in the first 250 evaluations, and slowed down in the rest of evaluations. This tells us that the BSO could find local optima quickly, but had difficulty finding high quality more globally optimal solutions. The final average score for BSO being only 887.0 means either we lost a Tank or seven Marines during the game. This is the lowest average score among the three tested algorithms. However, the best score obtained by the BSO is 1562 which shows that the quality of the best solution that BSO could find is relatively high. It destroyed all enemy units without losing a single unit. BSO's time score of 62 indicates the game last 38 seconds.

The RFO HC works slightly better than the BSO. Similar to BSO, it climbed fast in the first 250 evaluations and then slows down for the rest. However, the RFO found high quality solutions 7 out of 10 times with the same starting points as

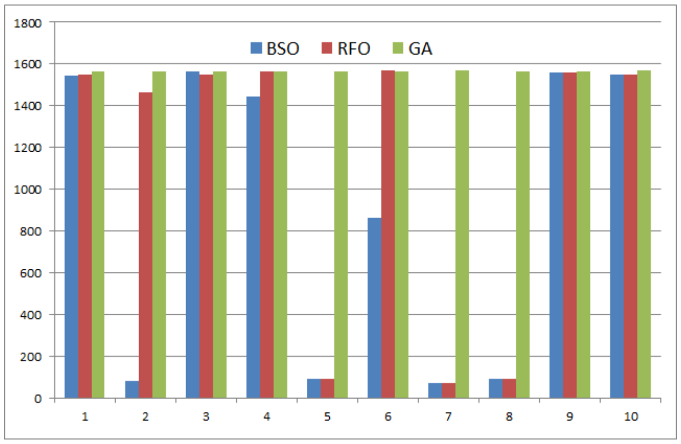


Fig. 4: Best scores of BSO, RFO, and GA with 10 different random seeds. X-axis represents random seed and Y-axis shows the highest fitness found by each algorithms initialized with each random seed.

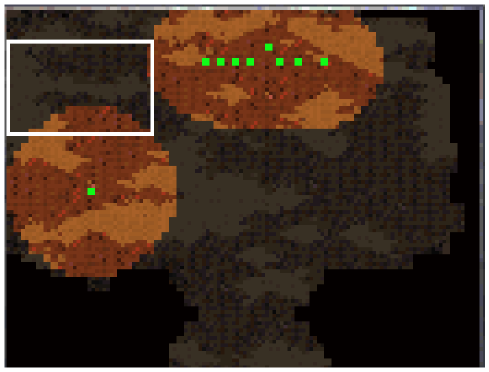


Fig. 5: Snapshot of one group positioning in Starcraft minimap. The dots on the map represent friendly units, and other part of the map was covered by fog of war. Single dot at the left of the map is Tank, and other dots are Marines.

BSO. It is more reliable than BSO based on average score as shown in Figure 3. Final average score is 1106.6 which is better than BSO. The best score found by RFO is 1567 which ended up with no unit lost and 5 seconds faster than the best score found by BSO.

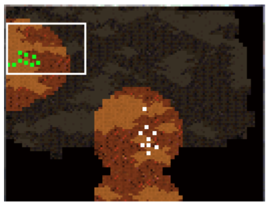
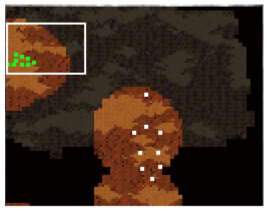
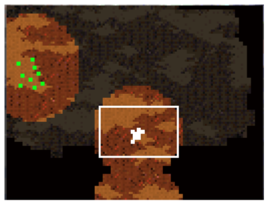
The GA was also applied in the experiments to compare the performance and quality with HCs. We used a population of 80 for maximum of 60 generations. The total number of evaluations was $80 \times 60 = 4800$. Figure 3 shows the average of maximum scores and average of average scores in each generation. The GA converged fast to 1500 during the first 20 generations which is $80 \times 20 = 1600$ evaluations, and then increased fitness slowly during the remaining 40 generations. The big difference between GA and HCs is that GA always (a hundred percent of the time) found good solutions. Also the average of the best scores converged to 1566, and the best score from the GA is 1567. This indicates that every run of the GA found high quality, near-optimal solutions, and those solutions are close to the best score possible (1600). Furthermore, the best solution from the GA is 5 seconds faster than RFO and 10 seconds than BSO. This indicates that the skirmish finishes in a short time and reduces uncertainties from opponent reactions and increases the safety of our own units.

From the point of view of IMs and PFs we can use IMs for guiding our units' positioning and move smoothly to attack the opponents units using PFs to guide our movement. Figure 5 shows an example of generated IMs positioning. The units could take advantage of this positioning to concentrate their fire and maximize their damage to the opponent. The group positioning and movement that evolves first learns to ensure that single units stay away from enemy unit controlled territory or to move outside of the map. If the enemy repulsor force is too small, units might move into enemy territory and be destroyed. On the other hand, if the force is too large, it will push the units to the border of the map and lead to avoiding the enemy altogether. Second, the parameters for the IMs were learned to guide our unit's positioning. The IM calculated the enemy's weak spots from the current position of enemy units and generates attraction points to guide our units in preparing for the skirmish. Different IM parameters lead to different locations, if the R_{Marine} and R_{Tank} are small, the locations might be inside the enemy units' attack range. If they are too large, the units may spend more time on the way and result in longer games, and low S_{time}. The enemy replusor and friend attractor were learned last. This affects detailed unit movement. Good combinations of attractors and replusors allow the group to move and attack smoothly and effectively. Units move to the right locations quickly and destroy enemy units faster. At the same time our units have more opportunity to survive. Therefore, our evaluation function is biased towards short movement, more enemy units eliminated, more own units survival, and shorter game duration.

We were also interested in the robustness of the solutions found by GAs and the two HCs from the point of view of the enemy's initial positioning. We wanted to know how our optimal solutions applied in different environments and were also curious how enemy initial position impacted our fitness scores. We designed three types of different custom maps in Starcraft in which the enemy were initially well dispersed, well concentrated, or in an intermediate position. The intermediate map was used for all prior results above. Table III specifies these initial enemy dispositions. These three types of scenarios can usually be found in human player matches. Dispersed units have less concentrating fire power but more map control and information gain. However, concentrated units have less map control but are harder to destroy.

We applied the solutions obtained from our initial intermediate scattered map to the two other maps. Each map was tested 500 times to get the average scores and their standard deviations. Table III shows these test results. The optimum was obtained from GA running in intermediate initial position and had the highest fitness of 1567. We tested this solution 500 times and on average obtained a fitness 1380.728 on the intermediate map. The standard deviation over all 500 tests on this map was 198.9 indicating the average error is within 2 Marines. We tested the same solution in a map with dispersed enemy units initial positions. The average fitness of 500 tests on this map was 1423.364. This is a higher score on a map never seen by the GA. The reason for the higher average score is that enemy units are dispersed and can be eliminated one

TABLE III: Average fitnesses and standard deviations of 500 matches on three maps with different initialized enemy units' position. Dots on the left side of the map represent the friendly units, and dots on the middle of the map represent the enemy units.

Enemy Initial Position	Description	Fitness
	Intermediate enemy position initialized, maximum distance from 2 units is 6 IM cells, which is also the default map for all the HCs and GAs experiments.	1380.7 $\sigma = 198.9$
	Dispersed enemy position initialized, maximum distance from 2 units is 11 IM cells. New scenario added to test robustness of the solution of previous map.	1423.4 $\sigma = 62.6$
	Concentrated enemy position initialized, maximum distance from 2 units is 3 IM cells. New scenario added to test robustness.	181.8 $\sigma = 346.1$

by one with very little damage. This tells us the optima we get from intermediate scattered position could work well or even better with more scattered enemy units. This also showed how group positioning is important in combat. The standard deviation of the tests on this map was low at 62.6. This matches our intuition that dispersed units are easily destroyed one by one quickly without much damage to the opponent, and our group with concentrated units has more concentrated fire power to damage the enemy. On the other hand, the average fitness of the tests on a map with concentrated units is low at 181.838. This means the matches were even on this map and only one or two units survived on average in 500 tests. The Marines and Tank were not able to synchronize their movement to confront enemy units at the same time, while concentrated enemy units could maximize their damage by firing at the same time. The standard deviation is 346.1, and is the highest on the three types of tests.

For comparison, Figure 6 shows the performance of the same CHC GA running on the intermediate scattered map and concentrated map. The graphs show that the enemy group with more scattered units is easier eliminated and faster for the GA to find quality solutions. It converged in the 20_{th} generation. However, the enemy group with more concentrated units got lower fitness because the enemy could damage opponent units more. The convergence rate is also slower than scattered units because the quality solution is harder to find.

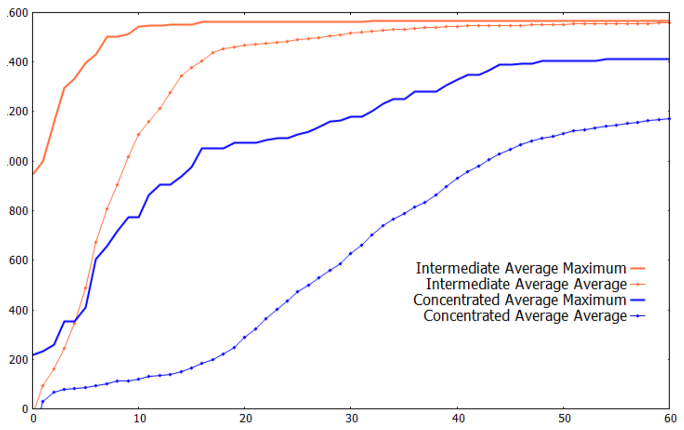


Fig. 6: Average maximum and average fitness of GA running on two types of map. X-axis represents generation, and Y-axis represents fitness.

V. Conclusion and Future Work

This paper compared GAs with two HCs to generate group positioning and unit movement based on influence maps and potential fields for beating opponents in a typical skirmish scenario in RTS games. We used Starcraft's built-in AI as our opponent baseline against which to make our comparisons. We compactly represented group behaviors in a combat as a combination of IMs and PFs parameters and restricted our search space to 2^{48}. We were able to compare GA performance versus much faster Bit-Setting Optimization and Random Flipping hill climbers. Results show that both BSO and RFO HCs can find local optima quickly against the baseline, but they are not guaranteed to find good solutions every time. They find good solutions between 50% to 70% of the time starting with ten different random seeds. Compared to HCs, the GA always find good combinations of IMs and PFs parameters and produce higher quality solutions compared to the hill-climbers. However, GAs take much longer to converge. Good solutions find good units attack positions, produce smooth unit movement, avoid unit collisions, synchronize attacking, and complete the skirmishes quickly.

We also compared the performance of GA running in different scenarios for testing robustness of the solutions found by GAs and HCs. The result shows that we could apply the solutions found in one scenario to more dispersed enemy units' position with high fitness. However, these solutions do not do well against more concentrated enemy positions.

These results apply to different aspect of designing a RTS game player. For example, with easy difficulty levels we can apply fast HCs to find not so good solutions. For harder difficulty levels, we would need to apply GA to find a better solution which need more computational time but provide much more challenge to players. Or we can mix two of the algorithms to increase the unpredictability and fun when playing against the AI. On the other hand, we can also allocate unbalanced computational and memory resource for different algorithms.

We are also interested in techniques which speed up finding high quality solution for skirmish. Some methods like case-injection or expert system may be added to our system in the future to increase solution finding performance. In addition, instead of evolving solutions based on a static baseline such as the built-in Starcraft AI, we could apply co-evolutionary techniques to produce both sides of the game AI.

Acknowledgment

This research is supported by ONR grant N00014-12-1-0860.

References

[1] D. Thomas, "New paradigms in artificial intelligence," *AI Game Programming Wisdom*, vol. 2, pp. 29–39, 2004.

[2] N. Wirth and M. Gallagher, "An influence map model for playing ms. pac-man," in *Computational Intelligence and Games, 2008. CIG '08. IEEE Symposium On*, dec. 2008, pp. 228 –233.

[3] (2005) Age of empires 3. [Online]. Available: www.ageofempires3.com

[4] 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.

[5] D. Aha, M. Molineaux, and M. Ponsen, "Learning to win: Case-based plan selection in a real-time strategy game," in *Case-Based Reasoning Research and Development*, ser. Lecture Notes in Computer Science, H. Muoz-vila and F. Ricci, Eds. Springer Berlin Heidelberg, 2005, vol. 3620, pp. 5–20.

[6] S. Ontañón, K. Mishra, N. Sugandh, and A. Ram, "Case-based planning and execution for real-time strategy games," in *Proceedings of the 7th international conference on Case-Based Reasoning: Case-Based Reasoning Research and Development*, ser. ICCBR '07. Berlin, Heidelberg: Springer-Verlag, 2007, pp. 164–178. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-74141-1_12

[7] S. Louis and C. Miles, "Playing to learn: case-injected genetic algorithms for learning to play computer games," *Evolutionary Computation, IEEE Transactions on*, vol. 9, no. 6, pp. 669 – 681, December 2005.

[8] 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

[9] 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.

[10] 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.

[11] 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.

[12] 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.

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

[14] S. Jang and S. 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.

[15] 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.

[16] 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.

[17] 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.

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

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

[20] 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

[21] M. Buro, "Orts a free software rts game engine," *Accessed March*, vol. 20, 2007.

[22] 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.

[23] B. Farkas and B. Entertainment, *StarCraft: Prima's official strategy guide*. Prima Communications, Inc., 2001.

[24] S. W. Wilson, "Ga-easy doe not imply steepest-ascent optimizable," 1991.

[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/