Using CIGAR 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 investigate using case-injected genetic algorithms to quickly generate high quality unit micro-management in 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 win even when outnumbered. In this paper, we use influence maps to generate group positioning and potential fields to guide unit movement and compare the performance of case-injected genetic algorithms, genetic algorithms, and two types of hill-climbing search in finding good unit behaviors for defeating the default Starcraft Brood Wars AI. Early results showed that our hill-climbers were quick but unreliable while the genetic algorithm was slow but reliably found quality solutions a hundred percent of the time. Case-injected genetic algorithms, on the other hand were designed to learn from experience to increase problem solving performance on similar problems. Preliminary results with case-injected genetic algorithms indicate that they find high quality results as reliable as genetic algorithms but up to twice as quickly on related maps.**

I. Introduction

The field of real-time strategy (RTS) game has become a popular focus of attention for artificial intelligence (AI) research in recent years. RTS games are a sub-genre of strategy computer and video games where players need to gather resources, build structures, research technologies, and conduct simulated warfare. Understanding each of these factors and their impact on decision making is critical for winning an RTS game. This paper focuses on applying Case Injected Genetic AlgoRithms (CIGARs) to generate competitive group positioning and movement reliably and quickly as part of building an RTS game player that can perform effective micro-management in a skirmish scenario. In the future, we plan to incorporate these results into the design of more complete RTS game players.

A typical RTS game may include several skirmishes and losing even 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. 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 win even when outnumbered. 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, and withdrawing seriously damaged units from combat. We focus on spatial positioning decisions involving the spatial shape of the battleground: locating the weaknesses of the enemy's defenses, selecting the targets to attack, and positioning units for an engagement.

Influence Maps (IMs) have been used for handling spatial problems in video games, robotics, and other areas [1]. 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 IM function, parameterized for each type of game entity - in this case enemy Marines. In our simulation environment, we set the weight of enemy units to be negative, therefore, lower cell values indicate more enemy Marines in the area and more danger to friendly 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 search algorithms to find near-optimal IM parameters that help specify high quality group positioning of our units for skirmishes on a battlefield.

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

While good influence maps can tell us where to go, good navigation can tell us how best to move there. We use Potential Fields (PFs) to control movement behaviors for a group of units navigating to particular locations on the map. PFs are a technique from robotics research for coordinating multiple units' movement behaviors and are often used in video games for the same purpose. A PF defines a vector force in space that attracts or repulses entities in a game. A well-known example of a potential field is gravitational potential. PFs have

978-1-4673-5311-3/13/$31.00 ©2013 IEEE

been applied to RTS games mostly for spatial navigation and collision avoidance. We apply PFs to coordinate units' group movement in our work.

Our ultimate goal is to create complete human-level RTS game players and this paper investigates one aspect of this problem: Finding good group positioning and movement for winning a skirmish scenario. Several challenges have to be handled in this research. First, how do we tune IM parameters for each type of unit to get good spatial information? Furthermore, PF parameters and IM parameters are interdependent if we use IM generated positioning information to guide PF mediated navigation. 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.

Early results showed that our hill-climbers quickly find influence maps and potential fields that generate quality positioning and movement in our simulations, but they only find quality solutions fifty to seventy percent of the time. Genetic Algorithms (GAs) on the other hand evolve high quality solutions a hundred percent of the time, but take significantly longer. Case-Injected Genetic AlgoRithms (CIGARs) combine GAs with case-based reasoning to learn to increase performance with experience. Since human players also learn to be better with practice and experience, we are interested in whether we can use CIGARs to learn from experience to be quicker and still get high quality solutions (like the GA) for winning skirmishes.

We consider a sequence of skirmishes as a sequence of problems to be solved by CIGAR and expect that CIGAR will learn on problems early in the sequence to improve performance on problems later in this sequence. Different skirmishes usually contain different unit compositions and positions. In this paper, we look into skirmishe scenarios with the same number of units but differently positioned. This enables CIGAR solutions obtained for one scenario to be injected into the evolving population of the next (similar) scenario and help augment the population with individuals that may contain useful information gleaned from prior scenarios. Specifically, we defined five similar scenarios with different enemy units' initial positions. We applied CIGARs and GAs to these five sequential scenarios and compared the performances of CIGARs and GAs on each scenario. Our preliminary results from this group of similar scenarios, show that CIGARs learn from prior experience to reliably find quality solutions on new scenarios in half the time taken by GAs. We expect that these results will further our research into generating good RTS game players.

The remainder of this paper is organized as follows. Section II describes related work in AI research and common techniques used in RTS games. Section III describes our simulation environment and parameter encodings for the GA and CIGAR. Section IV presents preliminary results and compares the quality, reliability, and the time taken to find solutions. Finally, section V draws conclusions and discusses future work.

II. Related Work

We first consider research in case-based reasoning 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 [2]. They built a case-base of previously encoded strategies, and the system learned to select, or index, the best matching strategy for different game states. They also performed an interesting analysis on the complexity of RTS games and showed that RTS games were difficult and thus suitable for computational intelligence research. Ontañón also worked on a real-time case based planning and execution techniques in RTS games and applied his technique to WARGUS [3]. 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 an 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 harder to scale.

Besides the research on case-based reasoning, we are also 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 [4]. 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 [5]. 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 [6]. 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 [7]. 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 opponents' positions [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. Uriarte applied influence maps for "kiting" units [9]. Raboin et al. presented a heuristic search technique for multi-agent pursuit-evasion games in partially observable space [10]. In this paper, we use three IMs to gather spatial information and guide our units for winning skirmish scenarios in RTS games.

Potential fields have also been applied to AI agents in RTS games. Most of this work is related to spatial navigation and collision avoidance [11]. 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 [12]. It was then widely used in avoiding obstacles and collisions especially for multiple unit flocking [13], [14], [15]. Hagelback brought this technique into AI research within the RTS game genre [16]. He presented a Multi-Agent Potential Field based bot architecture in ORTS [17]. Hagelback's paper applied potential fields at the tactical and unit operation level of the player AI [18]. We use three potential fields for group navigation in our work.

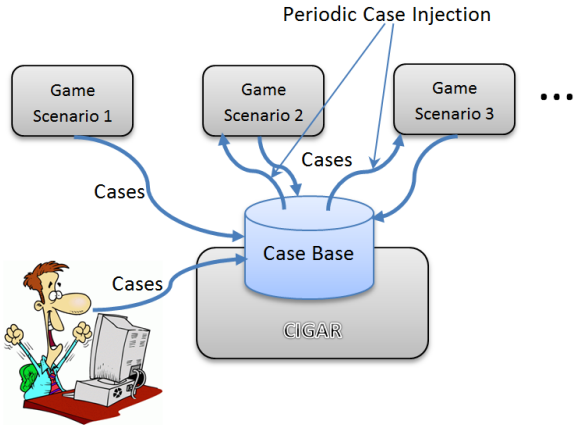

Fig. 2: Solving problems in sequence with CIGAR.

CIGAR was first introduced by Louis and McDonnell, they borrowed ideas from case-based reasoning (CBR) in which experience from solving previous problems helps solve new similar problems [19]. This approach augmented GAs with a case-based memory of past problem solving experience and was used to obtain better performance over time on sets of similar problems. Figure 2 shows how CIGAR solves a sequence of problems. An existing case-base is not necessary in case-injected GAs because the GA simply starts with a randomly initialized population to search the space. During such search, we save good chromosomes which are cases into the case-base for potential use in subsequent problem solving. Case-injection enable genetic algorithms to learn from experience. Louis and Li applied CIGAR for solving traveling salesman problems (TSPs) and showed performance improvement on similar TSPs [20]. Louis and Miles applied CIGAR in a strike force asset allocation game [21]. They used the cases from both human's and system's game-playing experiences to bias CIGAR toward producing plans that contain previous important strategic elements.

We are interested in techniques which can reliably and quickly find high quality solutions in skirmish scenarios in RTS games. Early results showed that hill-climbers are quick but unreliable while genetic algorithms are slow but reliably find good solutions. In this paper, we focus on applying CIGARs to speed up finding high quality solutions for skirmish scenarios.

III. Methodology

The first step of this research was building an infrastructure in which to run our AI agent in Starcraft: Brood War [22]. In our one versus one scenarios, each player controls the same number and types of units starting at two different places on the map. In order to compare with our earlier work in this area, we use the same rules which are listed below and the same customized maps.

- The maps do not have any obstacles.
- The units are default Starcraft units so there is no alteration of game balance.
- All the units use default Starcraft settings without upgrades.

TABLE I: Parameters defined in Starcraft

Parameter	Marine	Tank	Purpose
Hit-points	40	150	Entity's health. Hit-points decrease when hit by opponent's unit. Entity dies when Hit-points ≤ 0.
Size	12×20	32×32	Entity's size in pixel.
MaxSpeed	4.0	4.0	Maximum speed of Entity.
MaxDamage	6	30	Maximum number of Hit-points that can be subtracted from the target's health.
Range	128	224	The maximum distance at which an entity can fire upon target.
Cooldown	15	37	Time between weapons firing.
Score	100	700	Score gained by opponent when this unit has been eliminated.

- Units select targets based on Starcraft's built-in AI.
- There is no fog of war.

Our goal is to eliminate opponent units while minimizing the loss of friendly units. We also want to minimize 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. As before, our scenarios contain 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 more hit-points and attack range but costs more to produce. Table I shows the detailed properties for Marines and Tanks in Starcraft. In this preliminary work, we use the default Starcraft AI to control enemy units.

A. Influence Maps and Potential Fields

We compactly represent group behavior as a combination of three IMs and three PFs. Enemy unit generated influence maps tell our units where to go and our units navigate to the indicated positions using potential fields for movement. For the five preliminary scenarios considered in this paper, we only used three influence maps for enemy units. Specifically, an enemy Marine influence map, an enemy Tank influence map, and a Sum influence map derived from the previous two, that sums the enemy Marine and Tank influence maps. The enemy Marine and Tank influence maps were specified by the `weights` and `ranges` of enemy Marines and Tanks. Since computation time depends on the number of IM cells, we used a cell size of 64×64 pixels on the Starcraft map.

Potential field functions used in our experiments are 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.

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

We use three PFs of the form described by Equation 1 to control the movement of entities in a game. Each of these potential fields 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

TABLE II: Chromosome

Parameter	Bits	Description
W_{Marine}	5	Marine Weight in IMs
R_{Marine}	4	Marine Range in IMs
W_{Tank}	5	Tank Weight in IMs
R_{Tank}	4	Tank Range in IMs
c_a	6	Attractor Coefficient
c_{rf}	6	Friendly repulsor Coefficient
c_{re}	6	Enemy repulsor Coefficient
e_a	4	Attractor Exponent
e_{rf}	4	Friendly repulsor Exponent
e_{re}	4	Enemy repulsor Exponent
Total	48	

like $F = \frac{2000}{d^2}$. Here $c = 2000$ and $e = -2$ with respect to Equation 1.

- **Friend Repulsor:** This keeps friendly units moving towards a common goal from colliding with each other. It is typically stronger than the attractor at short distances and weaker at long distances. A typical repulsor 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 potential field is determined by two parameters, a coefficient and exponent of distance d, six parameters determine a unit's potential field:

$$PF = c_a d^{e_a} + c_{rf} d^{e_{rf}} + c_{re} d^{e_{re}} \quad (2)$$

where c_a and e_a are parameters of the attractor potential function, c_{rf} and e_{rf} for the friend repulsor, and c_{re} and e_{re} for the enemy replusor. These parameters are then encoded into a binary string which worked for both GAs and HCs, and is the chromosome of our GAs. Specifically, we encoded the chromosome in a 48-bit string. The detailed representation of IMs and PFs parameters are shown in table II. Note that the Sum IM is derived by summing the Marine and Tank IMs and so does not need to be encoded. When the game engine receives a chromosome, it decodes the binary string into corresponding parameters and directs friendly units to move and attack enemies according to Algorithm 1. The fitness of this chromosome at the end of each match was computed as described later and then sent back to our search algorithm.

Algorithm 1 Unit Navigation Algorithm

```
Initialize MarineIM, TankIM, SumIM;
while (checkAllUnitsInPosition() == false) do
    targetPosition = getMinimunPositionOnSumIM();
    PFMove(allUnits, targetPosition);
    if (unit in targetPosition) then
        setUnitInPosition(unit, targetPosition);
        updateIMs();
    end if
end while
enemyPosition = getEnemyPosition();
Attack(enemyPosition);
```

B. Fitness Evaluation

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

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

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 Marines 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 normal game speed. Therefore, time score being 0 means the game used up $T = 2500$ frames, a time score of 100 means that the game ended within one frame. We took game time into our evaluation because "timing" is an important factor in RTS game. Suppose a skirmish 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 skirmish thus increasing the chances of our player losing the skirmish. Therefore, skirmish duration becomes a crucial issue that we want to take it into consideration in our evaluation function.

However, especially during the early stages of evolution, if an individual cannot guide our units to engage the enemy units, F will be 0 and Equation 3 cannot then tell the difference between two such individuals. 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 scores. We used Equation 4 to evaluate matches when there is no engagement during the game.

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

where $\overline{D}$ is average distance from friendly units to enemy units. $1 - \frac{\overline{D}}{D_{max}}$ converts maximization of average distance to average 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. Genetic Algorithm

Our genetic algorithm used CHC elitist selection instead of selection used in canonical genetic algorithm [23], [24]. Offspring and parents together compete for population slots in the next generation in elitist selection. More specifically, our genetic algorithm selects the N best individuals from both parents and offspring to create the next generation. Early experiments indicated that our elitist selection genetic algorithm worked significantly better than the canonical genetic algorithm on our problem. According to previous experiments in our lab, we set the population size to 40 and ran the GA for 60 generations. The probability of crossover was 0.88 and we used CHC selection. We also used bit-mutation with a 0.01 chance that any individual bit would flip in value. Standard roulette wheel selection was used to select chromosomes for crossover. CHC being strongly elitist, helped keep valuable information from being lost if our GA produces low fitness children.

D. Case-Injected Genetic Algorithm (CIGAR)

The CIGAR used in this paper operates on the basis of hamming distance for solution similarity [19]. Therefore, our solution similarity distance metric is computed by the following formula:

$$D(A, B) = \sum_{i=0}^{l-1} (A_i \oplus B_i) \tag{5}$$

where l is the chromosome length, $\oplus$ represents the exclusive or operator (XOR), and A_i represents the ith bit of solution A.

We use a "closest to best" injection strategy to choose individuals from the case-base to be injected into CIGAR's population. We replace the four (10% of the population size) worst individuals with the individuals retrieved by our injection strategy. We chose the injection interval to be $log_2(N)$ where N is the population size. Therefore, we inject four "closest to best" cases every $log_2(40) \approx 6$ generations and replace the four worst individuals. We configured the population, selection, crossover and mutation in CIGAR to be the same as the GA.

E. Scenarios

To evaluate the performance of genetic algorithm with case injection and without case injection, we designed five different but similar scenarios. The difference between scenarios is the initial positions of enemy units'. Our five scenarios are:

- **Intermediate Position:** Enemy units located in the middle of the map. The maximum distance between any two enemy units is six IM cells.
- **Dispersed Position:** The maximum distance between any two enemy units is eleven IM cells. Enemy units in this scenario have the most scattered positions and the weakest concentrated fire power of all the five scenarios.
- **Concentrated Position:** Enemy units located in the middle of the map, and the maximum distance between any two enemy units is three IM cells. Enemy units in this scenario have the most concentrated fire power.
- **Corner Position:** Enemy units located at the northeast corner of the map, concentrated as much as in the previous scenario. Enemy units in this scenario have the strongest concentrated fire power as well as the best defensive positions.
- **Split Position:** Enemy units located in the middle of the map and split into two groups. Enemy units in this scenario have stronger concentrated fire power than Dispersed and Intermediate positions but weaker concentrated fire power than Concentrated and Corner positions.

These five types of scenario can usually be found in human player matches. Dispersed units have less concentrated fire power but more map control and information gain. On the other hand, concentrated units have less map control but are harder to destroy. Since our experiments do not have fog of war, the advantage of dispersed enemy units do not apply in this case resulting in making the more concentrated enemy units harder to destroy. Therefore, the first two scenarios and the last one are relatively easy for GAs to find high quality solutions to; scenario two and three are harder.

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 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. The randomness is used in ladder games as well. This however means that in our case, evolved parameters will not guarantee the same score every time we run. In other words, a high fitness solution has a high probability of getting a high score in the game.

According to the evaluation function and scenarios we introduced before, the theoretic maximum score for eliminating enemy forces is 1500 and maximum time score (corresponding to minimal time) is 100, therefore, the maximum evaluation score or fitness is 1600. Note that the first two digits in an evaluation score represent the unit elimination score, and the last two digits represent time score. For example, if the final score ends up at 1357 we can infer the following:

- The score being positive means our AI player defeated the built-in AI.
- 1300 represents the unit elimination score and compared to the maximum of 1500, our AI player lost 200 which indicates that two Marines were killed by the enemy during the game.
- The last two digits being 57 represents $(1 - \frac{57}{100}) \times 2500 = 1075$ frames spent during the entire game which is approximately 38.4 seconds converted to standard game speed.

A. *Case Injection's Effect on Genetic Algorithms*

In our experiments, we tested GAs and CIGARs running with ten different random seeds. Each such test took 14 hours to run the $40 \times 60 \times 5 = 12,000$ evaluations, where 40 is population size, 60 is the number of generations to run, and 5 represents the five scenarios. Scenarios are tested sequentially in the following order. Intermediate, Dispersed, Concentrated, Corner, and Split. CIGAR extracts the best individual in each generation and stores this individual into the case-base; duplicates are discarded. When running on a problem, suitable cases chosen according to the "closest to the best" strategy from the case-base are injected into CIGAR's population. Each case may contain useful information about the new search space and be a partial solution to the current problem and thus bias the genetic algorithm to take advantage of "good" genes found by previous search attempts. The number of cases in the case-base usually increases with the number of problems solved.

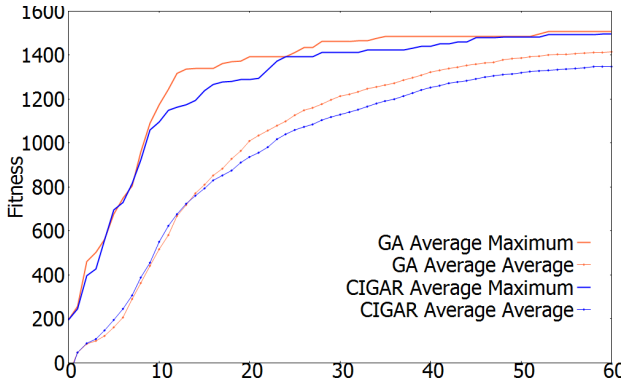

Fig. 3: Average maximum/average scores of GA and CIGAR over 10 runs on Intermediate scenario. The X-axis represents the generation and the Y-axis represents the fitness.

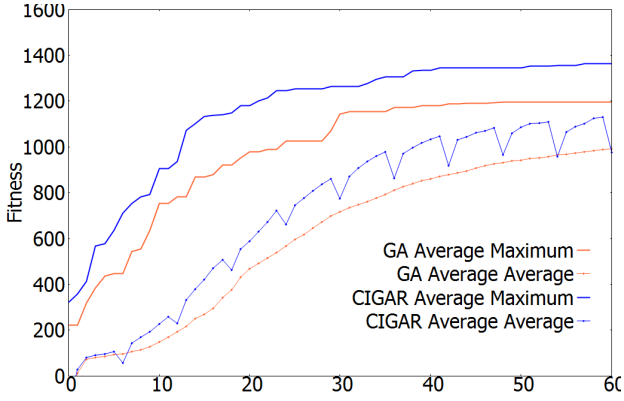

Fig. 4: Average maximum/average scores of GA and CIGAR over 10 runs on Concentrated scenario. X-axis represents the generation and Y-axis represents the fitness.

The performance of GAs and CIGARs running in the Intermediate scenarios as shown in Figure 3 tell us that their performance is similar. This is because the Intermediate scenario is the first problem to be attempted and CIGAR has no cases in its case-base when running on this problem. They are not exactly the same because there is some randomness in the game evaluation as explained earlier. On the other hand, the results from the Concentrated scenario show the difference in performance (see Figure 4). CIGAR has solved two problems (Intermediate and Dispersed scenarios) before this scenario, and 12.2 cases on average (over the ten runs) exist in the case-base. We can see that CIGAR outperformed the GA both in quality and speed in the Concentrated scenario when CIGAR's case-base contains cases from previous scenarios. The curve for CIGAR's average fitness in the Concentrated scenario dropped a little every 6 generations because four cases from the case-base were injected into the population. The new cases may only contain partial solutions with lower fitness in the population, which cause the average fitness to drop. However, average fitness rises again quickly after the drop. This shows the cases injected into the population may have introduced useful information leading to better performance.

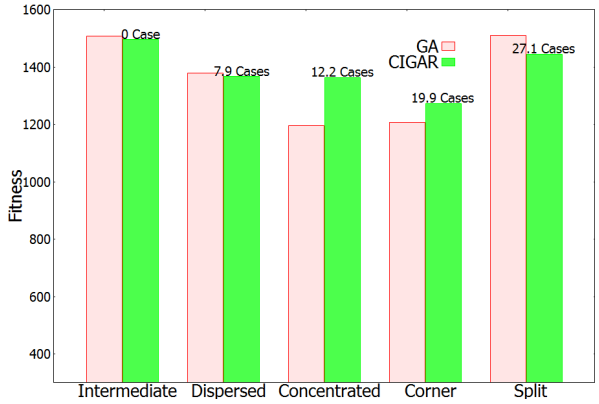

Fig. 5: Solution quality of each scenario. As more problems are solved, CIGAR produces better solutions than genetic algorithm. The X-axis represents 5 different scenarios. The Y-axis represents the highest fitness. The number on top of each bar of CIGAR shows the number of cases in case-base when CIGAR starts.

Figure 5 compares the quality of solutions found by the GA and CIGAR in all five scenarios. The number on top of each bar for CIGAR shows the average number of cases when CIGAR starts in the corresponding scenarios. The case-base is empty on the Intermediate (first) scenario, and increases to 27.1 on the last scenario. Therefore, CIGAR's "experience" generated from solving problems increases scenario by scenario. Both GA and CIGAR reliably found quality solutions above fitness 1200 a hundred percent of the time. The first two scenarios and the last one show that the GA and CIGAR found similar quality solutions. The reason behind this is that the Intermediate, Dispersed and Split scenarios are relatively easy to solve because scattered enemy units are easily destroyed one-by-one quickly without much damage to the opponent. Therefore, both GA and CIGAR performed very well. On the other hand, the Concentrated and Corner scenarios show the difference in performance between the GA and CIGAR. Concentrated enemy units have stronger fire power than scattered units which leads to high quality solutions being harder to find by GAs in the search space. In this case, we believe CIGAR had an advantage and case-injection biased search to more quickly find solutions with higher fitness.

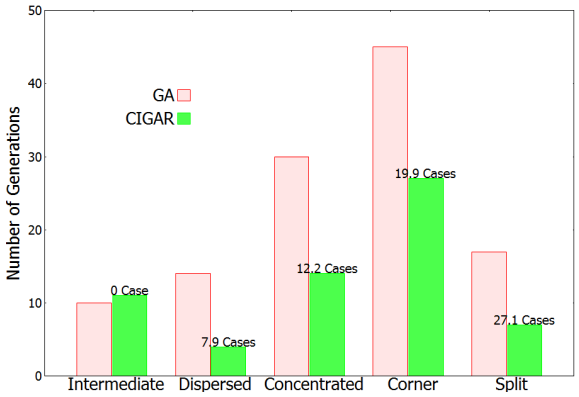

Fig. 6: Number of generations to solutions found above 1100. As more problems are solved, CIGAR took less time compared to the GA.

We wanted techniques for finding high quality solutions quickly and reliably for winning a skirmish in a RTS game. Thus, we measured the number of generations our GA and CIGAR took to produce quality solutions with fitness above a threshold quality of 1100. Figure 6 shows the number of generations needed to find quality solutions above 1100 from our GA and CIGAR. The GA ran on each scenario without any bias from injected cases and so GA performance can be used to indicate a level of difficulty for each scenario. A low number of generations indicates that the GA finds quality solutions easily. A high number of generations indicates the GA has a hard time finding quality solutions. So we can think of the GA performance in Figure 6 as showing us the difficulty levels of our five scenarios. In order from easy to hard, the scenarios are: Intermediate, Dispersed, Split, Concentrated, and Corner. The first scenario's result shows CIGAR found quality solutions 1 generation on average slower than our GA because CIGAR's case-base is empty. However, after CIGAR runs on the first scenario, on average 7.9 cases are stored into the case-base. CIGAR only takes 4 generations to find solutions with fitness greater than 1100 on this second scenario (Dispersed), while our GA takes 10 generations. Having found quality solutions for another scenario, the number of cases in our case-base increased again. CIGAR finds quality solutions for the third scenario in 14 generations compared to the GA which takes 30 generations. CIGAR found quality solutions 21 generations faster for the fourth scenario (the most difficult for the GA), and 10 generations faster for the last scenario.

B. Learned Group Behavior

From the influence maps and potential fields point of view, we use IMs to guide friendly units' positioning and use PFs to navigate friendly units to move and attack opponent units. Figure 7 shows an example of generated IMs positioning. Friendly 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 units 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 friendly units' positioning. The IMs calculated the enemy's weak spots from the current position of enemy units and generates attraction points to guide friendly 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 friendly units have more opportunity to survive. Therefore, our evaluation function is biased towards short movement, more enemy units eliminated, more friendly units survival, and shorter game duration.

Fig. 7: Snapshot of one group's positioning on Starcraft's minimap. The green dots on the map represent friendly units. The gray dots in the middle of the map are enemy units. The rest of the map is covered by fog of war.

Summarizing, the results indicate that CIGAR can produce high quality solutions for skirmishes reliably and quickly. CIGAR can learn through experience gained from previously attempted problems and bias search to reduce the time taken to find quality solutions to similar problems. In relatively easy scenarios like Intermediate, Dispersed and Split, the highest quality solutions found by GAs and CIGARs are close. However, CIGAR found high quality solutions with fitness above 1100 up to twice as fast as the GA. In the case of hard problems like Concentrated and Corner scenarios, CIGAR is not only two times as fast as the GA, but also found higher quality solutions than the GA. We are not ready yet to apply these results directly to the design of an RTS game player since we need to include health, terrain, and other aspects of the game world into our representation. Note however, that we can choose among different sets of skirmish parameters selecting the parameters from the scenario that best matches the current scenario.

V. Conclusion and Future Work

Our research focuses on generating group positioning and unit movement in order to win skirmish scenarios in RTS

game. Early results showed that hill-climbers can find serviceable IM and PF parameters that can occasionally defeat the default AI, but they are not guaranteed to find good solutions every time. Compared to hill-climbers, the genetic algorithms always find good combinations of IMs and PFs parameters and produce higher quality solutions, but take much longer to converge. Therefore, we are interested in techniques that can reliably and quickly find high quality solutions.

In this paper, we investigated applying case injected genetic algorithms to learn from "experiences" generated from previous problems and use this information to bias our search and speed up the process of finding high quality solutions. We defined five similar scenarios with different difficulty levels and you might expect to find similar scenarios in some human player matches. We applied CIGARs to these five sequential scenarios to assess how experience, stored as cases in a case-base affects performance compared to a GA. GAs with exactly the same parameters as used by CIGAR thus served as a baseline.

The results show that CIGARs performed similar to our GAs in the first scenario when the case-base is empty. In scenarios with more scattered enemy units, including scenarios one, two, and five, which are relatively easy problems, both GAs and CIGARs found high quality solutions. However, CIGARs find high quality solutions up to twice as fast as GAs. Finally, in scenarios two and three, with more concentrated enemy units, CIGARs not only find higher quality solutions than GAs, but also doubled the speed of finding a quality solution above 1100. This indicates that CIGARs are a suitable technique to apply across similar problems in RTS games. In addition, these "experiences" generated from solved problems provided valuable information and could help to speed up solving other similar problems.

We are also interested in applying CIGARs to more complicated scenarios where we consider different terrain, unit hit-points (health), weapon range, and other elements. 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. Furthermore, we want to evaluate our AI player against state-of-the-art bots from other researchers.

ACKNOWLEDGMENT

This research is supported by ONR grants N00014-12-1-0860 and N00014-12-C-0522.

REFERENCES

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

[2] 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., vol. 3620. Springer Berlin Heidelberg, 2005, pp. 5–20.

[3] 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

[4] 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.

[5] 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.

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

[7] 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.

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

[10] 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.

[11] 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.

[12] 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.

[13] 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.

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

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

[16] 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

[17] M. Buro, "Ortsa free software RTS game engine," *Accessed March*, vol. 20, p. 2007, 2007.

[18] 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.

[19] S. Louis and J. McDonnell, "Learning with case-injected genetic algorithms," *Evolutionary Computation, IEEE Transactions on*, vol. 8, no. 4, pp. 316–328, 2004.

[20] S. J. Louis and G. Li, "Case injected genetic algorithms for traveling salesman problems," *Information Sciences*, vol. 122, no. 2, pp. 201–225, 2000.

[21] 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, 2005.

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

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

[24] 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/