Co-evolving Real-Time Strategy Game Micro

Navin K Adhikari*, Sushil J. Louis† Siming Liu‡, and Walker Spurgeon§
Department of Computer Science and Engineering
University of Nevada, Reno
Email: *navinadhikari@nevada.unr.edu, †sushil@cse.unr.edu ‡simingl@unr.edu, and §wspurgeon@nevada.unr.edu

***Abstract*—We investigate competitive co-evolution of unit micromanagement in real-time strategy games. Although good long-term macro-strategy and good short-term unit micromanagement both impact real-time strategy games performance, this paper focuses on generating quality micro. Better micro, for example, can help players win skirmishes and battles even when outnumbered. Prior work has shown that we can evolve micro to beat a given opponent. We remove the need for a good opponent to evolve against by using competitive co-evolution to evolve high-quality micro for both sides from scratch. We first co-evolve micro to control a group of ranged units versus a group of melee units. We then move to co-evolve micro for a group of ranged and melee units versus a group of ranged and melee units. Results show that competitive co-evolution produces good quality micro and when combined with the well-known techniques of fitness sharing, shared sampling, and a hall of fame takes less time to produce better quality micro than simple co-evolution. We believe these results indicate the viability of co-evolutionary approaches for generating good unit micro-management.**

***Index Terms*—Co-evolutionary genetic algorithm, Influence map, Potential field, Real-time strategy game, Micro**

I. Introduction

Real-Time Strategy (RTS) games have become a new research frontier in the field of Artificial Intelligence (AI) as they represent a challenging environment for an autonomous agent. An RTS game player needs to collect resources, use those resources to construct a base, train units, research technologies and control different types of units to defeat the opponent while at the same time defending their own base from opponent attacks in a complex dynamic environment. All of these different actions that can be executed in any given state make a huge decision space for a player. RTS players usually divide these decision spaces into two different levels of tasks: macromanagement and micromanagement. Macromanagement encompasses a wide variety of tasks such as collecting more resources, constant unit production, technology upgrades, and scouting. In contrast, micromanagement is the ability of a player to control a group of units to beat an opponent. Better micro, for example, can help players win skirmishes and battles even when outnumbered or minimize damage received. Although good long-term macro-strategy and good short-term unit micromanagement both impact real-time strategy games performance, this paper focuses on generating quality micro. More specifically, we focus on two aspects of the micromanagement: tactics and reactive control [1]. Tactics deal with the overall positioning and movement of a group of units while reactive control deals with controlling a specific unit to achieve commonly used micro techniques: concentrating fire on a target, retreating seriously damaged units from the front line of the battle, and kiting (hit and run).

We build on prior work [2] and represent micro-behaviors of units on both sides with a set of parameters. We use a commonly used technique called Influence Maps (IMs) to represent enemy distribution over the game map. An IM is a grid placed over the map with a value assigned to each grid cell using an IM function that depends on the number of enemy units in the vicinity. Good IMs can tell us where the enemy is strongest and where they are weakest (That is, the best target position for friendly units to go). To navigate a group of units to a target location on the map given by an IM, we use Potential Fields (PFs). PFs are used widely in multi-agent systems for coordinated movement of multiple agents [3] [4]. We use two IMs parameters and four PF parameters for tactics and six parameters for reactive control. We then use a Co-evolutionary Genetic Algorithm (CGA) to search and find good combinations of these twelve parameters that lead to a good micro behavior on both sides.

Prior work has shown a Genetic Algorithm (GA) can evolve good micro to beat a given opponent but that micro performance depends on having a good opponent to play against. Furthermore, it is non-trivial to hard-code a good opponent to play against. We remove the need for a good opponent to evolve against by using competitive co-evolution to evolve high-quality micro for both sides from scratch. In competitive co-evolution, two populations evolve against each other. That is, individuals in one population play against individuals in the other population for evaluating each other's fitness. Note that the fitness of an individual in one population depends on the fitness of the opponents drawn from the other population. As both population evolve, individuals from each population must compete against more and more challenging opponents leading to an arms race. This simple model of co-evolution suffers from several well known problems [5]. Although even this simple model of co-evolution works well enough to produce better than random micro, we use three techniques: competitive fitness sharing, shared sampling and hall of fame from Rosin and Belew [6] to produce better quality micro in less time than using simple co-evolution.

We first co-evolve micro to control a group of ranged units versus a group of melee units. We then move to co-evolve micro for a group of ranged and melee units versus an opponent group of ranged and melee units. Results show that we can co-evolve good micro for both opponents in both scenarios. In addition, we tested generalizability of the co-

evolved micro by evaluating performance of co-evolved micro in different initial configurations and different initial positions of units. Results show that micro co-evolved in one scenario work well in other scenarios as well.

The remainder of this paper is organized as follows. Section II describes related work in RTS AI research, generating game players using co-evolutionary techniques and common techniques used in RTS micro. The next section describes our RTS research environment. Section III explains our CGA implementation. Section IV presents preliminary results. Finally, the last section provides conclusions and discusses possible directions for future work.

II. Related Work

Traditionally, much work has been done in Computational Intelligence (CI) and AI in games revolving around board games, using a variety of techniques [7] [8]. More recently, research has shifted away from board game towards more complex computer games and real-time strategy games like starcraft pose several challenges for computational intelligence research [9]. In addition, challenges in RTS games are strikingly similar to real-world challenges making RTS games a good research platform.

Much work has been done on RTS games addressing different aspects of developing an AI player for such games [1]. In this paper, we are interested in one aspect of an RTS game: "Micro." Micro stands for micromanagement, the reactive and tactical control of a group of units to maximize their effectiveness (usually) in combat. Game tree search techniques and machine learning approach have been explored for micro tactics and reactive control. Churchill and Buro explored a game tree search algorithm for tactical battles in RTS games [10]. Synnaeve and Bessiere applied bayesian modeling to inverse fusion of the sensory inputs of the units for integration of tactical goals directly in micro-management [11]. Wender and Watson evaluated different reinforcement learning for micromagement [12].

A number of micro techniques use influence maps and potential fields for tactical and reactive control of units in an RTS game. An Influence Map (IM) tiles a map into square tiles with each tile or grid cell getting a value provided by an IM function. Grid cell values determine enemy unit locations and concentrations and can be used to provide a variety of useful information for unit maneuvering. We describe influence maps and potential fields later in this paper. Miles evolved the parameters of an influence map using a genetic algorithm in order to evolve an RTS game player [13]. Sweetser and Wiles used IMs to help a decision-making agent in their EmerGEnt game [14]. Bergsma and Spronck generated adaptive AI for a turn-based strategy game using IMs [15]. Jan and Cho used information provided by layered IMs to evolve nonplayer characters' strategies in the strategy game Conqueror [16]. Preuss investigated flocking-based and IM-based path-finding algorithms to optimize group movement in the RTS game [17]. Uriarte and Ontanon used an IM-based approach to evolve kiting (similar to hit-and-run) behavior in the Starcraft bot Nova [18]. Danielsiek investigated influence maps to support flanking the opponent in a RTS game [19]. This paper uses influence maps to determine a target location to move towards and attack. Potential fields guide our unit movement.

Potential fields (PFs) of the form cd^e where d is distance and c and e are tunable parameters have been used in robotics and games for generating smooth group movement [20]. Relevent to RTS games, Jonas and Kostler used PFs to control units optimally in StarCraft II for simulating optimal fights [21]. Sabdberg and Togelius Hagel investigated multi agent potential field based AI approach for small scale combat in RTS games [22]. Rathe and Svendsen did unit micromanagement in Starcraft using potential fields [23]. They all used genetic algorithm to tune multiple potential fields' parameters. Hagelback and Johansson applied potential fields in their RTS games research [24]. They proposed a multiagent PF-based bot architecture for the RTS games ORTS and applied PFs for tactical and reactive unit movement. Closer to our work, Liu and Louis used parameterized algorithms that determined unit positioning, movement, target selection, kiting, and fleeing. Then a genetic algorithm tuned these parameters by evolving against a hand-coded opponent or an existing Starcraft BWAPI bot [2]. We build on this prior work and use the same representation (parameterized algorithms) but co-evolve, rather then evolve, micro without the need for a good opponent to play against.

Coevolution in games goes back to Shannon's work on checkers in the 50s with the most recent notable example being Alpha-go Zero [25] [26]. In RTS games, Ballinger and Louis showed that coevolution led to more robust build orders. Build-order optimization enables players to generate the right mix and numbers of units meeting a strategic need [27]. Avery and Louis coevolved team-tactics using a set of IMs, navigating a group of friendly units to move and attack enemy units on the basis of the opponent's position [28]. More relevant to our coevolutionary approach, Rosin and Belew improved the performance of a coevolution using three techniques: competitive fitness sharing, shared sampling and hall of fame [6]. We use these techniques and show their effectiveness in coevolving good micro in less time than simple coevolution.

The next section describes our game engine and provides details on how we simulate skirmishes in this game engine for fitness evaluation. We then describe our representation and evolutionary algorithm tuned parameters and our methodology for measuring coevolutionary progress.

III. Methodology

Apart from the Starcraft BWAPI and Starcraft II API, there now exist a number of other RTS game-like engines that can be used for RTS game research [29] [30]. The open-source FastEcslent game engine which runs game graphics in a separate thread is especially suitable for evolutionary computing research in games since we can run the underlying game simulation without graphics and thus more easily do multiple parallel evaluations. Figure 1 shows a screenshot from FastEcslsent running with graphics.

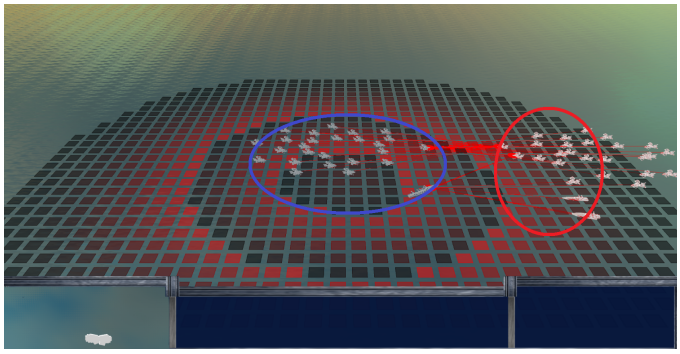

Fig. 1: Screenshot of a skirmish in FastEcslent

Parameter	Variable	Bits	Description
IM	W_U	5	Enemy unit weight in IMs.
	R_U	4	Enemy unit influence range in IMs.
PF	c_a	6	Attractor coefficient.
	c_f	6	Repulsor coefficient.
	e_a	4	Attractor exponent.
	e_f	4	Repulsor exponent.
	S_t	4	The waiting time to move after each firing.
Reactive control	D_k	5	The distance from the target that unit starts 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 in number of cells, for 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 units, under which unit will flee.
Total		51	

Fig. 2: Parameters tuned by coevolution

We use unit health, weapons, and speed values from Starcraft to create the equivalent of Vultures and Zealots in FastEcslent [31] [29]. A Vulture is StarCraft unit that is fast but fragile and can attack form a longer distance which helps such units "kite," during a skirmish while a Zealot is slower but stronger and has a shorter attack distance. To evaluate the fitness of a chromosome, we decode the chromosome and use the twelve resulting parameters to control our units in the game simulation. The simulation ends when all the units on one side are destroyed or time runs out. After each simulation, FastEcslent returns a score for each side in the skirmish based on how much damage was done and how much damage was received and this score is used to compute a fitness.

The goal of our work is to evolve good micro for opponents in an RTS game without the need of an opponent to evolve against. We therefore use a coevolutionary algorithm to achieve this goal. In coevolution, two populations of individuals play each other to compute fitnesses that drive evolution [6]. Extending prior work, we represent micro by a set of parameterized algorithms and the genetic or coevolutionary algorithm tunes these parameters to find good micro. These algorithms specify group positioning, movement, target selection, kiting, and fleeing. Table 2 details the twelve parameters in our representation which is identical to the representation used by Liu [2]. Tuning these parameters results in micro for one type of friendly unit against one type of enemy unit. We explain these parameters below.

Good positioning during a skirmish can reduce damage received and increase damage dealt to opponent units. We use Influence Maps (IMs) to try and find vulnerable positions to attack. An influence map is a grid of cells placed over the map, where each cell has a value determined by an IM function. In our work, the IM function specifies a weight parameter (W_e) for each cell occupied by an enemy entity. The entitys influence decreases as a function of distance and ceases after R_e distance (in number of cells) units. We sum the influence of all enemy entities in range of cell to compute the cells IM value. The cell with the lowest value and that is closest to the enemy determines our attack location. The GA or coevolutionary algorithm determines W_e and R_e.

How a group of units moves to the target location also determines skirmish performance. We use attractive and repulsive potential fields to control group movement [32]. The typical representation of an attractive and a repulsive potential field is given by equation 1

$$\mathcal{P}F = c_a d^{e_a} + c_r d^{e_r} \tag{1}$$

where C_a and E_a are parameters of the attractive force and C_r and E_r are parameters of repulsive force. However, balancing these forces to achieve smooth, effective unit movement is difficult and we therefore use the CGA to find the best values for these parameters. Once we reach the target location, target selection, kiting, and fleeing become important. Good target selection can significantly affect performance since selecting a weaker target, to destroy more quickly, can thus more quickly reduce damage being received. The CGA evolves the two parameters, HP_ef and R_nt, defined in Table 2 to guide a target selection algorithm [2].

Kiting by longer ranged units is an effective tactic used by good human players in skirmishes with short ranged melee units. Three parameters that determine kiting behavior are 1) how far away from a target the unit needs to start kiting (D_k), 2) the waiting time before moving after each firing (s_t), and 3) how far a unit should retreat before attacking (D_kb). We use a parameterized kiting algorithm which uses these three parameters to kite [2]. Finally, removing weakened units from the front line to save them for later is determined by a hit-point threshold HP_fb, also coevolved by the CGA. Good values for all these parameters can lead to micro that beats state of the art BWAPI competition bot micro [9] [2] when evolved against such micro. This paper seeks to use CGAs to reach

high levels of performance without the need for good micro to evolve against.

A. Coevolution and Fitness Evaluation

In coevolution, individual fitnesses result from direct competition between individuals in two populations. We want to maximize damage done and minimize damage received. More precisely, when an individual i, from one population competes against individuals from the other population, i gets a score given by equation 2, based on damage done by N_f friendly units to N_e enemy units and damage received in each competition.

$$\begin{aligned} Score &= V_1 \sum_{n=1}^{N_f} (HP_f / HPF_{max}) \\ &+ V_2 \sum_{n=1}^{N_e} (HPE_{max} - HP_e) \end{aligned} \tag{2}$$

HPF_{max} is the starting hitpoints corresponding to maximum health for each friendly unit. Similary HPE_{max} specifies the starting hitpoints of each enemy unit. HP_f represents the remaining hitpoints for friendly units at the end of a fitness simulation while HP_e represents the same parameter for enemy units. V_1 and V_2 are scores for saving friendly hitpoints (health) or reducing enemy hitpoints. We obtain these values from the Starcraft BWAPI. We explain how this score leads to an individual's fitness after describing the coevolutionary algorithm.

Since we are coevolving both sides, we refer to the two sides coevolving in their distinct populations as red and blue. Figure 3 shows how individuals in the blue and red populations are evaluated and how a single evaluation determines the fitness of two individuals - one from the blue and one from the red population.

B. One unit type versus one unit type

Our first experimental scenario coevolved 5 red Vultures against 25 blue Zealots. For each individual in the blue population, we send the 12 parameters specified by that individual to control micro for the 25 blue side zealots against every individual in the red population. Each red individual's chromosome controls the 5 vultures. For a population size, p, and assuming both red and blue have the same population size, we need a total of p^2 evaluations to obtain a fitness for every individual in both populations.

Equation 2 specified the score received during one evaluation; an individual's fitness is the average of all the scores obtained by playing one red individual (for example) against all p members of the blue population. V_1 and V_2 differ for the red and blue populations since each is trying to micro a different type of unit. $V_1 = 400, V_2 = 160, HPF_{max} = 80, HPE_{max} = 160, N_e = 25$ (zealots), and $N_f = 5$ vultures for the red population. From the blue population's point of view, these parameter values are different. Blue friend Zealots compete against red enemy Vultures and $V_1 = 160, V_2 = 80, HPF_{max} = 160, HPE_{max} = 80, N_e = 5$ (vultures), and $N_f = 25$ zealots for the blue population. Except for the

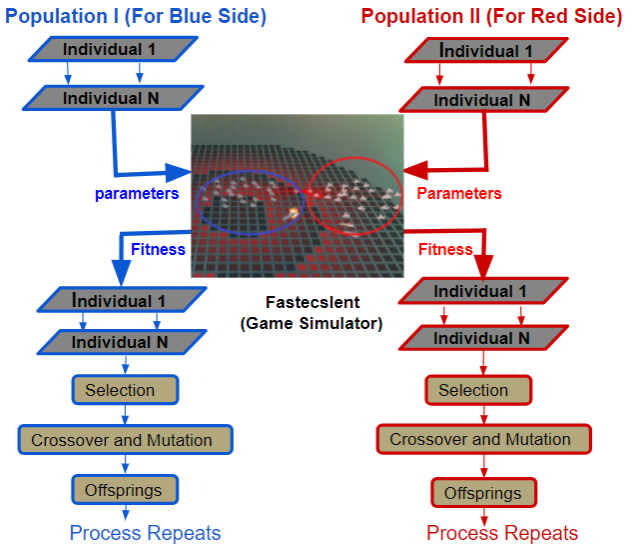

Fig. 3: The coevolutionary algorithm plays individuals from the blue population against individuals in the red population to obtain damage done and received and thus determine relative fitness.

number of enemies, N_e, and number of friends, N_f, the values of all other parameters are obtained from the Starcraft1.

C. Two unit types versus two unit types

Good results from coevolving micro for groups composed from one type of unit versus groups also composed from one, albeit differnt, type of unit led us to consider a second set of experiments where we investigated coevolving micro for groups composed from two types of units against an opponent group also composed from two types of units. Specifically, we coevolved micro for a group of 5 vultures and 25, say on the red side against an identical group of 5 vultures and 25 zealots on the blue side. Our chromosomes doubled in size from 12 to 24 parameters and the first 12 parameters controlled vultures while the second set of 12 parameters controlled zealots.

We also generalized Equation 2 to handle multiple types of friend and enemy units. Essentially this means that there are two values for V_1, one for vultures (400) and one for zealots (160). Similarly there are two values for V_2 when considering damage to enemy vultures (80) and zealots (160). Maximum values for hitpoints also depend on the unit type.

For simple competitive coevolution, the fitness of an individual is the average of scores obtained from playing against all individuals in the opponent population. An individual plays against another by being placed in our game and running the game until either all the units in one side are destroyed or time runs out. Once we have such a measure of fitness, the two populations can potentially coevolve leading to an arms-race of increasing fitness.

Although this model of coevolution works well enough to produce better than random micro, we use three techniques: competitive fitness sharing, shared sampling, and hall of fame

as described by Rosin and Belew [6] to produce better quality micro in less time than using simple coevolution [33]. We provide brief descriptions of these three methods below.

The idea of fitness sharing is to prevent diverse niches from prematurely going extinct. Sharing an individuals score from defeating a specific individual i drawn from the opponent population among all the individuals that defeated i, leads to higher fitness for individuals that defeat opponent individuals that no one else can. This decreases the probability of important innovations going extinct.

The usual way to evaluate an individual is to play against all the individuals in the opponent population. To reduce computational effort, shared sampling evaluates an individual by playing against a sample of individuals drawn from the opponent population. In order to increase the diversity in this opponent sample, first select an opponent individual A that defeated the most individuals in your population. Then an individual defeating those individuals that defeated A are selected, and so on, until the sample size becomes full.

A finite population means that a high fitness individual from one generation may not stay high fitness in a different context provided by an evolving opponent population. To ensure against permanent loss of such strong individuals and to prevent cycling caused by intransitive superiority, we keep such current strong individuals in a hall of fame so that we can use a strong diverse sample of past (hall of fame) and current individuals to play against in order to gain a better measure of an individuals fitness. This helps evolve individuals that are more robust.

IV. Results and Discussion

For all evaluations, we ran for a maximum of 2500 frames which, despite running without graphics, took an average of 5 seconds per evaluation. We therefore parallelized evaluations to get reasonable run times and achieved approximately linear speedup. Coevolution run results are averaged over ten runs with different random seed.

First, we coevolved micro for a group of ranged units versus a group of melee units with simple coevolution - that is coevolution without any shared sampling, shared fitness, or hall of fame. Second, we compared the results produced using simple coevolution with the results produced when using all three techniques.

Specifically, in the first set of experiments we coevolve 5 red vultures versus 25 blue zealots. Both populations used a population size of 50 and we ran for 60 generations. Crossover and mutation probabilities were 0.95 and 0.03 respectively.

Since the fitness of an individual, i, depends on the quality of individuals from the opponent population that i competes against in coevolution, plotting fitness over time does not measure progress in the same that such plots do for standard genetic algorithms. Instead, we use a different approach and start by generating a baseline individual. Every coevolutionary generation, we take the best individual from the blue (or red) population and play this best individual against the fixed baseline. As coevolution progresses, we expect the best individual in subsequent generations to improve performance over the fixed baseline.

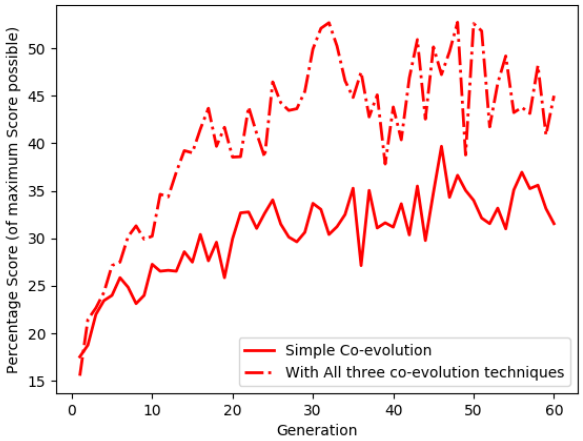

Fig. 4: Performance of coevolving vultures against the baseline zealots

Figure 4 plots the best coevolving vulture (red population) against such a baseline zealot. This baseline zealot beats $1996/2000$ randomly generated individuals for a 90% win rate. The solid line shows simple coevolution while the dashed line shows coevolution augmented with fitness sharing, shared sampling, and hall of fame. We can see improvement over time for both and we can see that the three techniques do improve micro quality faster. To reduce the computational effort, the size of shared sample and hall of fame should be as low as possible. But, decreasing the size too much may reduce needed diversity in the set of individuals selected for playing against. We thus need a delicate balance between maintaining diversity in the shared sample and hall of fame to play against, and low computational effort. In these and subsequent experiments the shared sample size and hall of fame size are both set to five (5) - a value found through experimentation. With these settings we get 50 population size $\times 10$, shared sampling size plus hall of fame size, $\times 2$ for the two populations for a total of 1000 evaluations per coevolutionary generation. This equates to a savings of $\frac{2500-1000}{2500} = 60\%$ in terms of computational effort measured in number of evaluations.

Figure 5 shows a similar patterns when comparing the coevolving zealots against a baseline vulture micro. Note that in both sets of results using the three methods result in smoother performance curves. Videos of gameplay show complex patterns of movement. Zealots learn to herd vultures into a corner while vultures learn kiting and to stay out of range of zealots. These videos are available at https://www.cse.unr.edu/ navin/coevolution.

Building on these results, we next investigated more complex micro for groups composed from zealots and vultures versus an opponent also built from the same two types of units. With two types of units we can also look to see whether, and

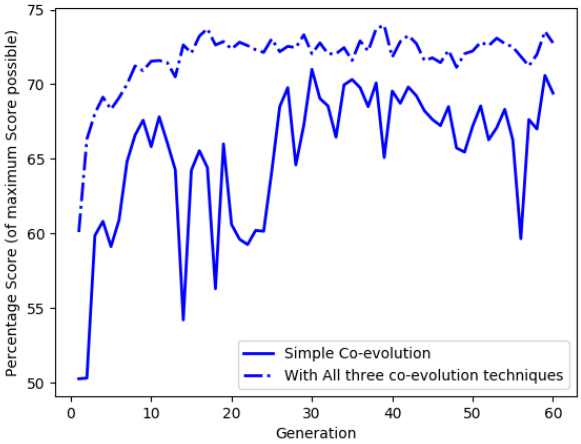

Fig. 5: Performance of coevolving zealots against the baseline vultures

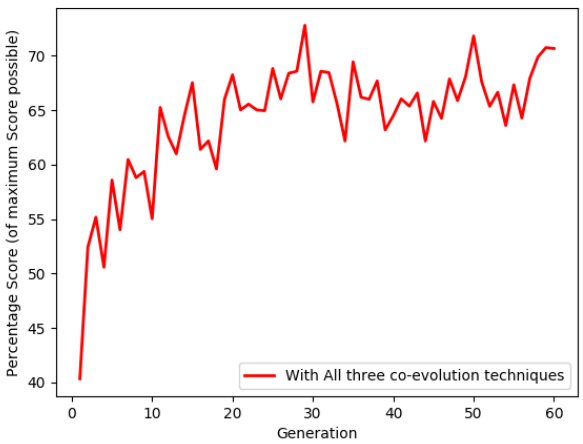

Fig. 6: Performance of coevolving zealots and vultures versus the baseline for the Red population

what kind of, cooperative behavior emerges between the two types of units.

A. *Two types of units versus two types of units*

We investigate coevolving micro for a group of 5 vultures and 25 zealots versus an identical opponent group (5 zealots, 25 vultures). We did not test simple coevolution, preferring to use the coevolution with the three methods since the augmented coevolutionary algorithm performs better and due to a lack of time.

Again, we used a population size of 50 running for 60 generations with the same crossover and mutation probabilities of 0.95 and 0.03 as before. Note that the chromosome size needs to double so that micro for the two unit types coevolve to take advantage of each unit type's unique properties. Progress is again measured against a baseline group of 5 vultures and 25 zealots that beats 99% of randomly generated opponents.

Figure 6 shows coevolutionary progress against this baseline for the red population. Again we see fairly smooth (for coevolution) progress in finding increasingly good micro. Figure 7 shows, unsurprisingly, that the coevolving blue population has similar performance improvement. These results seem to indicate the potential for a coevolutionary approach to coevolve good micro from scratch.

Next we consider the robustness of this coevolved micro by testing the coevolved micro in scenarios hitherto unseen. First, we looked at the micro coevolved for the one unit type versus one unit type experiments and selected the best co-evolved individual from both populations. We then played these two best individuals in three different starting formations (or scenarios) and ten different starting locations. We did the same for the best individuals in the two unit types versus two unit types experiments.

Figure 8 shows screenshots of these three scenarios. The first distributes units within a circle (labeled 1), the second uses a line formation (2), and the third distributes units

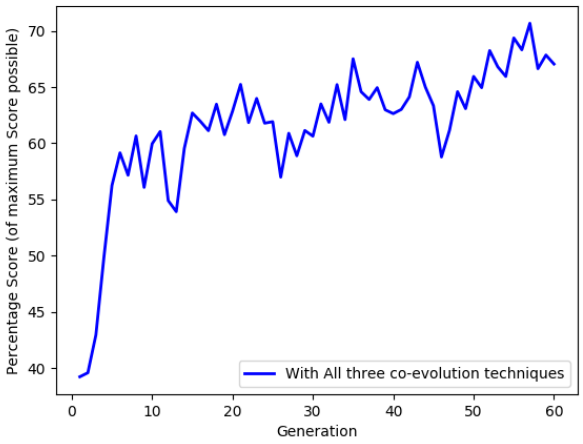

Fig. 7: Performance of coevolving zealots and vultures versus the baseline for the Blue population

randomly (3). Blue and Red indicate side in the screenshot. We describe our experiments and results with respect to these three formations next. Red bars represent red population (vulture) performance versus baseline zealots and blue bars represent blue (zealot) performance versus baseline zealots.

B. *Scenario 1: Circular*

This was our training scenario in that coevolution took place with units placed within this circle and always started in the same initial positions during a fitness evaluation. To test robustness we randomly changed the starting positions and generated 10 randomly generated sets of starting positions for the units. We tested the coevolved micro against our baseline player on these 10 different scenarios and computed the average score. Figure 9 shows that coevolved micro seems

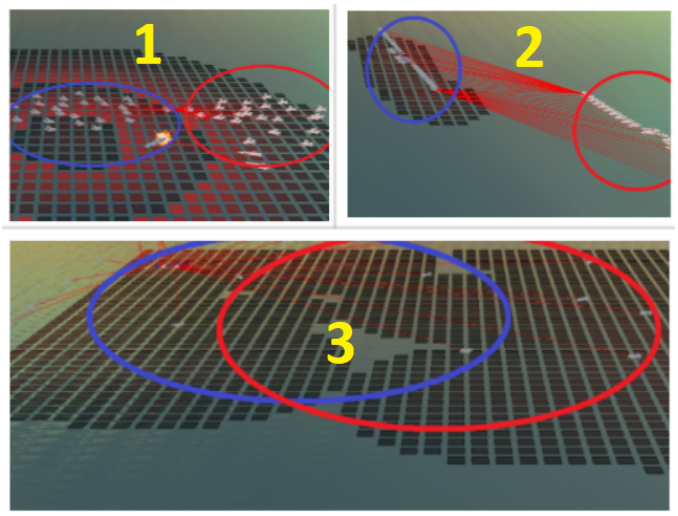

Fig. 8: Snapshot of circular formation (1), line formation (2), and random formation (3)

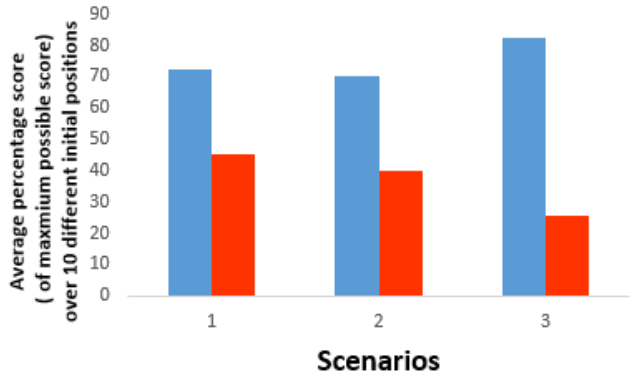

Fig. 9: Performance of co-evolved player against baseline in different scenarios; Co-evolved Vulture (Red), Co-evolved Zealot (Blue)

robust to starting position with performance similar to those in Figure 4 and Figure 5.

C. Scenario 2: Line formation

In this scenario, units from both sides area placed in a line opposite each other on the game map. With this formation, we want to see how the coevolved micro does when changing both the formation and the initial positions on this formation. Again, we randomly generated 10 different sets of unit starting positions on the line and averaged the score obtained by the best coevolved micro against our baseline. The second set of bars in Figure 9 shows that co-evolved micro does just as well in this new formation over multiple sets of starting locations.

D. Scenario 3: Random starting locations

In this scenario, rather than putting units into any particular formation, we place them randomly in the game. The score of the best individual against baseline player is again averaged

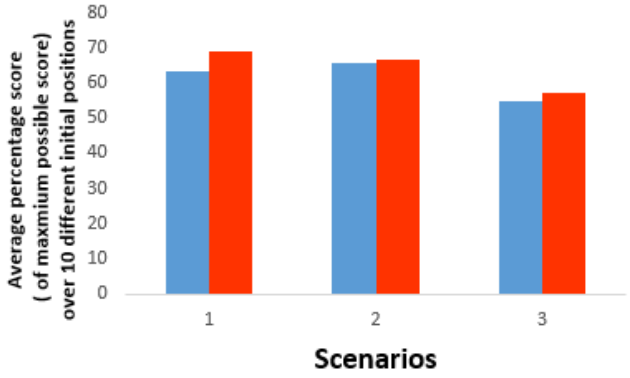

Fig. 10: Performance of co-evolved player against baseline in different scenarios; Red and Blue sides

over 10 different initial position and shown in Figure 9. With this formation, we can see that vultures do not fare well. We address this in our future work.

Finally, Figure 10 shows the same information for the two unit types versus two unit types experiments. We can see the same trend. These figures indicate the potential for our CGA approach to find good robust micro from scratch.

V. Conclusions and Future Work

RTS games pose several challenges for computational intelligence research. Having good micro in RTS games can help players win skirmishes and battles even when outnumbered. One of the solution techniques that are effective at searching what are typically vast search spaces in prior work is representing micro using a set of parameters and using GA to find the good combination of these parameters. But the performance of the micro obtained using GA depends on having a good opponent, which is non-trival to hard-code, to play against. In this paper, we presented a coevolutionary approach to finding good micro in RTS games which eliminates the need for a good opponent to evolve against.

We compactly represented micro with 12 parameters on both sides that control predefined algorithms for target selection, kiting, and unit movement and used a CGA to tunes these parameters values. Results show that we can coevolve a group of ranged units versus a group of melee units using simple coevolution. We also compare these results using three different techniques for improving competitive coevolution as described by Rosin and Belew [6]. Results also indicates that we can coevolve a better micro in less time than using simple coevolution.

We then use our approach to coevolve micro for units composed from heterogenous (two) types of units versus similar opponents. For a mix of ranged and melee units results show that we can coevolve good micro-behavior using coevolution augmented with shared sampling, fall of fame, and shared fitness.

We checked the robustness of our co-evolved micro in three different unseen scenarios and ten different sets of starting

positions. Results shows that our approach can find micro that performs well in unseen scenarios. We believe, using a combination of random and structured scenarios during coevolution will lead to more robustness.

The main constraints with a coevolution in an RTS game is computational effort for evaluations. As a single fight simulation takes significant time and each individual needs multiple evaluations, required computational effort tends to outstrip available resources. Although shared sampling and hall of fame result in good reduction, it still takes days to get significant results. Given more computational resources, we may be able to use much larger population sizes and much longer run times to get significantly higher quality.

We believe these results indicate the viability of coevolutionary approaches for generating good unit micromanagement and we plan to build on this in our future work. Although this work dealt one unit type and two unit types on bothsides, we would like to extend our work to multiple unit types. Furthermore, we plan to coevolve within Starcraft II using the recently released Starcraft II API to implement our approach and representation and test against human experts.

REFERENCES

[1] S. Ontaon, G. Synnaeve, A. Uriarte, F. Richoux, D. Churchill, and M. Preuss, "A survey of real-time strategy game AI research and competition in starcraft," in *IEEE Trans. Comput. Intell. AI Games,vol.5,no.4*, Dec 2013, pp. 1–19.

[2] S. Liu, S. J. Louis, and C. A. Ballinger, "Evolving effective microbehavior in real-time strategy games," in *IEEE Trans. Comput. Intell. AI Games,vol.8,no.4*, Dec 2016.

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

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

[5] J. Schonfeld, S. J. Louis, and J. Doe, "A survery of coevolutionary algorithms research," *evolution*, vol. 32, no. 67, p. 63.

[6] C. D. Rosin and R. K. Belew, "New methods for competitive coevolution," in *Evolutionary computation, 1997 - MIT Press*, 1997, pp. 1–29.

[7] J. Furnkranz, "Machine learning in games: A survey," in *Machine that Learn to Play Games*, Huntington, NY, USA: Nova Science, 2001, pp. 11–59.

[8] J. Rubin and I. Watson, "Computer poker: A review," in *Artif. Intell., vol.175,no.5*, 2011, pp. 958–987.

[9] M. Buro, "Real-time strategy games: A new AI research challenge," in *IJCAI, 2003 - skatgame.net*, 2003.

[10] D. Churchill, A. Saffidine, and M. Buro, "Fast heuristic search for RTS game combat scenarios," in *AIIDE*, 2012.

[11] G. Synnaeve and P. Bessiere, "A bayesian model for RTS units control applied to starcraft," in *Proceedings of IEEE CIG*, 2011.

[12] S. Wender and I. Watson, "Applying reinforcement learning to small scale combat in the real-time strategy game starcraft:broodwar," in *IEEE CIG*, 2012.

[13] C. Miles, J. Quiroz, R. Leigh, and S. Louis, "Co-evolving influence map tree based strategy game players," in *Proc. IEEE symp. Comput. Intell. Game*, April 2007, pp. 88–95.

[14] P. 6th Int. Conf. Intell. Data Eng. Autom. Learning, "Combining influence maps and cellular automata for reactive game agents," in *Proc. IEEE symp. Comput. Intell. Game*, 2005, pp. 209–215.

[15] M.Bergsma and P. Spronck, "Adaptive spatial reasoning for turn-based strategy games," in *Proc. Artif. Intell. Interactive Digital Entertain. Conf.*, 2008.

[16] S.-H. Jang and S.-B. Cho, "Evolving neural NPCs with layered influence map in the real-time simulation game conqueror," in *Proc. IEEE Symp. Comput. Intell. Games*, Dec 2008, pp. 385–385.

[17] 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," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 2, no. 2, pp. 82–98, 2010.

[18] A. Uriarte and S. Ontann, "Kiting in RTS games using influence maps," in *Proc. 8th Artif. Intell. Interactive Digital Entertain. Conf.*, 2012.

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

[20] O. khatib, "Real-time obstacle avoidance for manipulators and mobile rotobs," in *Int. J. Robot. Res. vol.5,no.1*, Jun 1986, pp. 90–98.

[21] J. Schmitt and H. Kstler, "A multi-objective genetic algorithm for simulating optimal fights in starcraft II," in *IEEE CIG*, 2016.

[22] T. W. Sandberg and J. Togelius, "Evolutionary multi-agent potential field based AI approach for SSC scenarios in RTS games," in *PHD thesis*, 2011.

[23] E. A. Rathe and rgen Be Svendsen, "Micromanagement in starcraft using potential fields tuned with a multi- objective genetic algorithm," in *Maste thesis*, 2012.

[24] J. Hagelback and S. Johansson, "Using multi-agent potential fields in real-time strategy games games games games," in *Proc. Artif. Intell. Interactive Digital Entertainment Conf.*, 2008.

[25] C.E.Shannon, "Game playing machines," in *Journal of the Franklin Institute Vol. 260 no. 6*, 1995.

[26] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot *et al.*, "Mastering the game of go with deep neural networks and tree search," *nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[27] C. Ballingers and S. Louis, "Comparing coevolution, genetic algorithms, and hill-climbers for finding real-time strategy game plans," in *GECCO '13 Companion Proceedings of the 15th annual conference companion on Genetic and evolutionary computation*, July 2013, pp. 47–48.

[28] P. Avery and S. Louis, "Coevolving influence maps for spatial team tactics in a RTS game," in *Proc. 12th -Annu. Conf. Genetic Evol. Comput.*, 2010, pp. 783–790.

[29] A. Heinermann *et al.*, "Bwapian api for interacting with starcraft: Broodwar," *available from (accessed 2014-02-03)*, 2012.

[30] Starcraft II API. [Online]. Available: http://us.battle.net/sc2/en/blog/20944009/the-starcraft-ii-api-has-arrived-8-9-2017

[31] An application programming interface for interacting with starcraft: Broodwar. [Online]. Available: https://bwapi.github.io

[32] P. Vadakkepat, K. C. Tan, and W. Ming-Liang, "Evolutionary artificial potential fields and their application in real time robot path planning," in *Evolutionary Computation, 2000. Proceedings of the 2000 Congress on*, vol. 1. IEEE, 2000, pp. 256–263.

[33] W. D. Hillis, "Co-evolving parasites improve simulated evolution as an optimization procedure," *Physica D: Nonlinear Phenomena*, vol. 42, no. 1-3, pp. 228–234, 1990.