Co-evolving Real-Time Strategy Game Playing Influence Map Trees With Genetic Algorithms

Chris Miles
Evolutionary Computing Systems Lab
Dept. of Computer Science and Engineering
University of Nevada, Reno
miles@cse.unr.edu

Sushil J. Louis
Evolutionary Computing Systems Lab
Dept. of Computer Science and Engineering
University of Nevada, Reno
sushil@cse.unr.edu

***Abstract*— We investigate the use of genetic algorithms to play real-time computer strategy games and focus on solving the complex spatial reasoning problems found within these games. To overcome the knowledge acquisition bottleneck found in using traditional expert systems, scripts, and decision trees as done in most game AI, we use genetic algorithms to evolve game players. The spatial decision makers in these game players use influence maps as a basic building block, from which they construct and evolve influence map trees containing complex game playing strategies. With co-evolution we attain ”arms race” like progress, leading to the evolution of robust players superior to their hand-coded counterparts.**

I. Introduction

Gaming and entertainment drive research in graphics, modeling and many other computer fields. Although AI research has in the past been interested in games like checkers and chess [1], [2], [3], [4], [5], popular computer games like Starcraft and Counter-Strike are very different and have not received much attention from researchers. These games are situated in a virtual world, involve both long-term and reactive planning, and provide an immersive, fun experience. At the same time, we can pose many training, planning, and scientific problems as games where player decisions determine the final solution.

Developers of computer players (game AI) for these games tend to utilize finite state machines, rule-based systems, or other such knowledge intensive approaches. These approaches work well - at least until a human player learns their habits and weaknesses - but require significant player and developer resources to create and tune to play competently. Development of game AI therefore suffers from the knowledge acquisition bottleneck well known to AI researchers.

By using evolutionary techniques to create game players we aim to overcome these bottlenecks and produce players which can learn and adapt. The games we are interested in are Real Time Strategy (RTS) games. These are games such as Starcraft, Dawn of War, Supreme Ruler (Figure 1), or Age of Empires [6], [7], [8], [9]. Players are given cities, armies, buildings, and abstract resources - money, gold, saltpeter. They play by both allocating these resources, to produce more units and buildings, and by assigning objectives and commands to their units. Units carry out player orders automatically, and the game is usually resolved with the destruction of other player’s assets.

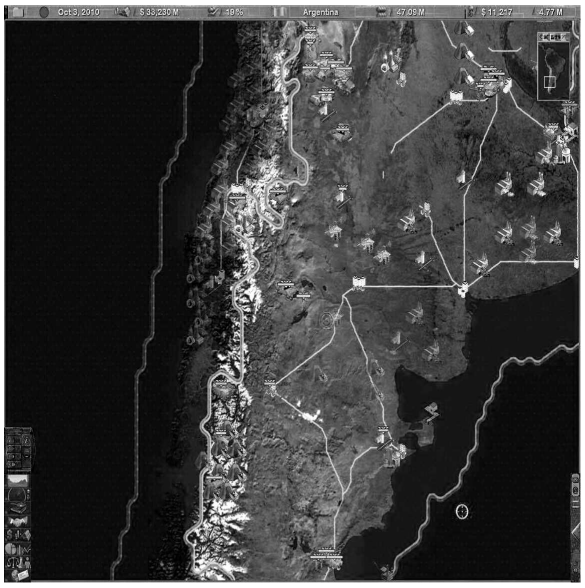


Fig. 1. Supreme Ruler 2010

Games are fundamentally about making decisions and exercising skills.

> ”A good game is a series of interesting decisions. The decisions must be both frequent and meaningful.” - Sid Meier

RTS games concentrate player involvement primarily around making decisions, in contrast to some genres such as racing games which require a high degree of skill. While varying greatly in content and play, RTS games share common foundational decisions. Most of these decisions can be categorized as either resource allocation problems: how much money to invest on improving my economy, which troops to field, or what technological enhancements to research; or as spatial reasoning problems: which parts of the world should I try to control, how should I assault this defensive installation, or how do I outmaneuver my opponent in this battle.

Our overall goal is to evolve systems to play RTS games, making both resource allocation and spatial reasoning decisions. Previous work has used genetic algorithms to make allocation decisions within RTS games [10]. For this research we explore the possibilities of evolving spatial decision makers, a key component of RTS game AI. These spatial decision making systems would be responsible for looking at the game world and deciding to build a base here, to put a wall up there, to send a feigning attack over there, and then to lay siege to that city. RTS games have, by design, a non-linear search space of potential strategies, with players making interesting and complex decisions which often have difficult to predict consequences later in the game. Using genetic algorithms we aim to explore this unknown and non-linear search space. We represent spatial decision making strategies within the individuals of a genetic algorithms population. The game theoretic meaning for strategy is used here - a system which can choose an action in response to any situation [11]. Then we develop a fitness function which evaluates these decision makers based upon their in-game performance - many games already tabulate a players performance automatically. A genetic algorithm then evolves increasingly effective players against whatever opponents are available. Due to the number of games and evaluations required to reach competent players we first use hand-coded automated opponents for this phase of the research. Co-evolution is the natural extension of playing against hand-coded opponents, whereby we evolve players against each other, with the goal of increasing game playing competence and strategic complexity.

In this paper we develop a spatial decision making system within the context of a 3D computer RTS game. We describe the game within which we test the system, evolving players first against static hand-coded opponents and later against another population of co-evolving players. Results present analysis of the genetic algorithms performance, including the behaviors produced by the system. Finally we discuss directions for the continuation of this research, and future work.

II. Representation

Each individual in the population represents a game-playing strategy. We use influence maps to represent spatial features. Influence maps evolved out of work done at spatial reasoning within the game of Go [12] and have been used sporadically since then in games such as Age of Empires [9].

A. Influence Maps

An influence map (IM) is a grid placed over the world, which has values assigned to each square based on some function which represents a spatial feature or concept. The IM function could be a summation of the natural resources present in that square, the distance to the closest enemy, or the number of friendly units in the vicinity. Figure 2 is a visualization of an influence map, on the left you have the game world which has two triangle units in it. On the right is an influence map, where the IM function is based upon distance to triangles. The highest valued points are those which are the closest to triangles, which are brighter in the visualization.

Several IM's are created and then combined to form the spatial decision making system. For example create two influence maps, the first using an IM function which produces high values near vulnerable enemies, the second IM function producing high negative values near powerful enemies. Then combine those two influence maps via a weighted sum. High valued points in the IM resulting from the summation, are good places to attack - places where you can strike vulnerable enemies while avoiding powerful ones. The final step is to analyze the resultant IM and translate it into orders which can be assigned to units. In this example we take the highest valued point, and tell our troops to attack there.

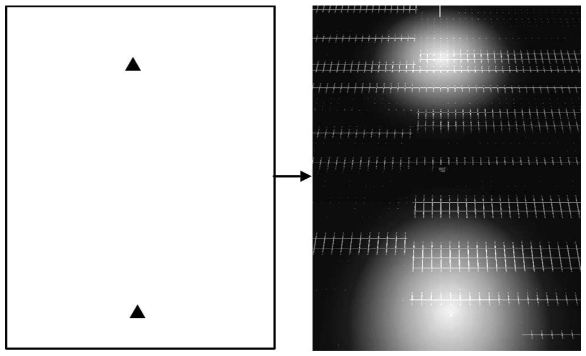


Fig. 2. Influence Map - Left $->$ Game World, Right $->$ Influence Map representing proximity to the Triangles

The set of IM functions and their parameters be applied to produce answers for any situation, so they can encapsulate a decision making strategy. Each IM conveys simple concepts: near, away, hide, attack; which combine together to form complicated behavior - hide near neutral units until your enemy is nearby then attack. In our work we encode the IM functions and their parameters within the individuals of a genetic algorithm, which we then evolve with standard genetic operators. Previous work [13] evolved a neural network which took every square from every IM as an input, and produced the squares of the final IM as output. Our system has the flexibility to evolve both the influence maps and their final combination, and since the combination operators are simple arithmetic operators the system is more transparent and therefor easier to analyze.

B. Influence Map Combinations

To allow for more generalization we combine IM's within a tree structure instead of the traditional list [12]. Each tree represents a complete decision making strategy, and is encoded within the individuals of a genetic algorithm. Leaf nodes in the tree are regular IM's, they use functions to generate their values based on the game-state as before. Branch nodes perform operations upon their children's values in order to create their own values. The kinds of processing performed by the branch node include standard arithmetic operators, such as a weighted sum, or multiplication, as well as more complex processing such as smoothing or normalization functions.

Influence map trees are a generalization of the traditional method of using a weighted sum on a list of influence maps [12]. They also allow for the variety of specialized processing done on influence maps in many commercial games.

For example, Age of Empires uses multi-pass smoothing on influence maps to determine where to construct buildings. IMTrees were designed to contain all the important information about influence maps within one structure. The IMTree structure can then be encoded as an individual in a population, including 1) the structure of the tree, 2) which IM functions to apply at each node, 3) which parameters to use in those functions, and 4) any processing to be done. With crossover and mutation operators we can then evolve towards more effective spatial decision making strategies. This is in many ways similar to genetic programming, but taken in the context of spatial reasoning. Next, we explore the effectiveness of this system in the context of a naval combat game - Lagoon.

III. The Game - Lagoon

We developed Lagoon, a Real-Time 3D naval combat simulation game. Figure 3 shows a screen-shot from the bridge of one destroyer which is about to collide with another destroyer. The world is accurately modeled, and the game can be played from either the helm of a single boat or as a real-time strategy game with players commanding fleets of boats. The complexities of the physics model are particularly demanding on the players, as the largest boats take several minutes to come to a complete stop. To deal with these and other complexities, Lagoon has a hierarchical AI system which distributes the work. At the top level sits the strategic planning system being developed by our group, this system allocates resources and assigns objectives to the various groups of boats. Behavior networks then carry out those orders for each individual boat, following proper naval procedure within the complexities and constraints of the physics model. They then relay their desired speeds and headings to a helmsman controller, which manipulates the various actuators and effectors on the boats - rudders and rpm settings to the engines.

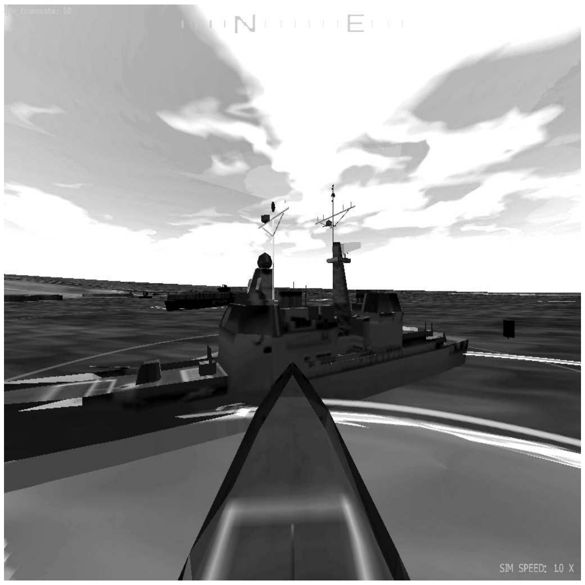


Fig. 3. Lagoon

A. The Mission

To test influence map trees we created the mission shown in Figure 4. Two small cigarette boats - triangles, attempt to attack an oil platform - pentagon, which is being guarded by a destroyer - hexagon. The cigarette boats are fast and maneuverable, and equipped with rocket propelled grenade launchers. Their primary advantage over the defending destroyer is that they can quickly accelerate, decelerate and turn. The destroyer on the other hand is quite fast, with a higher top speed then the attacking cigarette boats, but it takes a significant period of time to change speeds or turn. The six-inch gun on the destroyer has been disabled for this mission, requiring it to rely upon machine gun banks mounted on its sides.

This mission was chosen as it was relatively simple, and it required the players to understand the effectiveness of their units, with the attacker being able to coordinate a group attack. We also chose this mission because we could develop hand-coded players for both sides easily. In many ways this is more of a tactical than a strategic mission, in that there are few boats on each side, and no ”complex long term” decisions to make such as where to place a base. We think of this mission as an initial test of our ability to evolve effective spatial decision making strategies. Future work would be tested on missions involving large numbers of boats and more complex interactions.

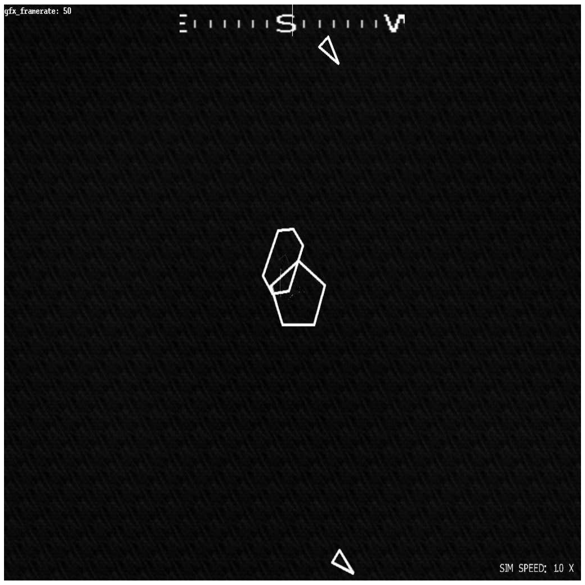
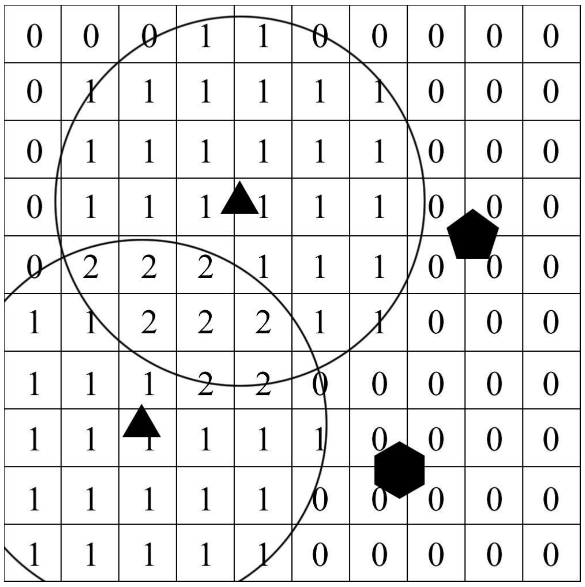


Fig. 4. Mission

B. Influence Map Tree Implementation

Each unit in the game - the two cigarette boats and the destroyer, have an influence map tree instantiated and assigned to them. The two attackers have duplicate IMTrees, so we are evolving a single attacking strategy for all situations. The IM’s calculate the value of their squares with IM functions based on which units are near those squares as shown in Figure 5. Units in the world add various circles of influence to each IM

- increasing the values assigned to all squares around those units. The IM function must first determine which units it considers relevant, this is based on a parameter which we encode in the GA. It can be either the unit the IMTree is assigned to, other friendly units, neutral units, or enemy units. The next issue, and GA parameter, is how large of a circle to use, with the IM either using the weapons radius of the unit it is assigned too, the weapons radius of the unit the circle is around, or a large fixed radius. Next, the IM determines how much to increment values within the circle. Each unit has an abstract power or strength rating associated with it, which gives a general idea how powerful that unit is in combat. The IM can either use this strength rating, or it can use the value of that unit. The next issue is how to distribute values within the circle. In Figure 5 we increased the value of each square within a circle by one, regardless of its distance to the unit the circle is centered around. The IM function can also distribute values with a bias towards the center, so points near the center get the maximum value and as you move towards the perimeter you get less and less points. There is an also an inverse distribution, giving maximum points at the perimeter and zero points in the center. All of these options to the IM function are parameterized, and encoded within individuals in the genetic algorithm. To allow fine tuning of each IM, two coefficient parameters are also encoded. The first directly scales the radius of the circle used for each unit - this is bound within (0,4]. The second directly scales the values given to squares within the circle - bound within [-10,10].



0	0	0	1	1	0	0	0	0	0
0	1	1	1	1	1	1	0	0	0
0	1	1	1	1	1	1	0	0	0
0	1	1	1	1	1	1	0	0	0
0	2	2	2	1	1	1	0	0	0
1	1	2	2	2	1	1	0	0	0
1	1	1	2	2	0	0	0	0	0
1	1	1	1	1	1	0	0	0	0
1	1	1	1	1	0	0	0	0	0
1	1	1	1	1	0	0	0	0	0

Fig. 5. Influence Map Implementation

Branch nodes in the IMTree can be any of the four basic operators - addition, subtraction, multiplication, and division. There is also an "OR" branch node which takes the largest values from its children at each point. The OR node generally functions as the root of the tree, choosing between the various courses of actions contained within its children. With these nodes we then constructed players for both sides, tuning and testing them over a few games.

Our hand-coded attackers work by using an OR node on two child subtrees. The first subtree represents an attack behavior which takes the weighted sum of three nodes. The first node has high values near vulnerable enemies, the second has large negative values near powerful enemies, and the third has negative values near other friendly units, to keep them from bunching up. The second subtree represents a run away behavior, which the cigarette boats should use if the destroyer gets too close, it works by multiplying an IM with high values away from the destroyer, and an IM with high values near the assigned unit. The defender counters this with a similar tree, once again using an OR node on two subtrees. The first behavior puts the destroyer in-between any attackers and the oil platform, it works by multiplying high values near valuable friendly units with high values near powerful enemies. The second behavior keeps the destroyer near the oil platform in the direction facing the attackers if it has nothing else to do, it is a multiplication of high values in close proximity to the oil platform, with high values in a very large are around the attacker. Both of these IM trees worked reasonably well, with the attackers trying to out-maneuver the defender while the defender diligently defends the oil platform.

We found that our hand-coded attackers were easily defeated by our hand-coded defender. The defender was effective, staying near the oil platform until the cigarette boats approached. The destroyer would then put itself between the cigarette boats and the oil platform, blocking them from getting too close. If they continued to approach it would fire upon them and destroy them. The cigarette boats would try to out-maneuver the destroyer, taking advantage of their maneuverability to get to the far side of the oil platform where they thought they could attack with impunity. They had a hard time getting from one side of the destroyer to the other however, often entering its field of fire and being destroyed. To improve upon this we turned to evolutionary techniques, allowing the GA to evolve IMTrees for controlling units in our game.

IV. Evolution

We evolved our players with a non-generational genetic algorithm with roulette wheel selection, one point crossover and bitwise mutation. Crossover took place with 75% probability, and the bitwise mutation probability was chosen to give on average two bit mutations per child. At this initial phase we were not evolving the structure of the tree, purely the parameters and coefficients for each IM. The GA uses the same structure as our hand-coded attackers and defenders. More complicated missions and strategies would likely require a more complex tree, but we found this structure to be sufficient for our desired behavior.

A. Encoding

The GA packs all the parameters for each IM in the IMTree into a bit-string, with fixed point binary integer encoding for the enumerations and fixed point binary fraction encoding for the real valued parameters and coefficients.

B. Evaluation and Fitness

To evaluate each individual we play them against an opponent and examine the results of the match. Fitness is calculated as $fitness = damagedone - damagereceived$ at the end of the game, which makes it a zero-sum two player game.

V. Results

We originally expected the attacker to develop an attack-distract strategy, with one cigarette boat distracting or occupying the destroyer while the other cigarette boat attacks the oil platform. We expected the defender to loiter around the oil platform, chasing off any attackers which come nearby. We found our hand-coded attackers were relatively ineffective against the defender. So we first evolved the attackers IMTree against our hand-coded defender. We analyze the behaviors produced by the resultant IMTree, and comment upon its effectiveness. Then we evolved the defenders IMTree against the evolved attacker. We analyze the behaviors it produces, commenting upon how they have reacted to the attacker. Finally we evolve the two populations simultaneously, describing the behavior within the system.

A. Results: Evolving the Attacker

The GA evolved our attackers IMTree against the hand-coded defender, using a bit-string of length 198 bits. The GA evaluated 1000 individuals against our hand-coded defender. We graph the fitness of the best, worst, and average individual in the population after each evaluation in Figure 6. While we ran the system multiple times, we will discuss a single representative run which illustrates the results we consistently achieved. The 205th attacker found a strategy for which the defender had no effective response, this strategy is shown in Figure 7. The attackers make a quick pass on oil platform from the far side of the destroyer before it can reach an effective cruising speed, and then sail off into the sunset. The defender patrols around the oil platform afterward, but the attackers never attempt a second pass. The attackers weapon does good damage, but has a very long reload time, so the single pass attack does significant damage without presenting risk to the attackers. The destroyer starts at rest and takes a long period of time to reach its maximum speed. This gives the attackers enough time to make it to the far side and attack while it is still accelerating. Once the destroyer has picked up speed it is difficult for the attackers to avoid being intercepted before they can get within range of the oil platform. Therefor it makes sense for the attackers to do what damage can be done easily, and then run away. To test if this behavior was dependent on the initial mission layout, we randomized the location of the attackers and still consistently evolved this behavior.

B. Results: Evolving the Defender

The evolved attackers were effective against the hand-coded defender, doing damage and fleeing before retaliation could be enacted. We next re-ran the genetic algorithm to evolve the defenders IMTree, to see if it could find a counter to the attackers strategy. The fitness of individuals in the population are shown in Figure 8. The defender cannot stop the attackers from making their initial pass. What it did do was learn to duplicate the attackers run-away behavior. The defender does not learn to follow the attackers, only to run off in the same direction they run off in by duplicating that part of their tree. In doing so it abandons the oil platform, leaving it totally vulnerable to any other attacking strategy. However, as the destroyer has a higher top speed than the cigarette boats, it eventually chases them down and annihilates them. This over specialized behavior was effective against the attackers

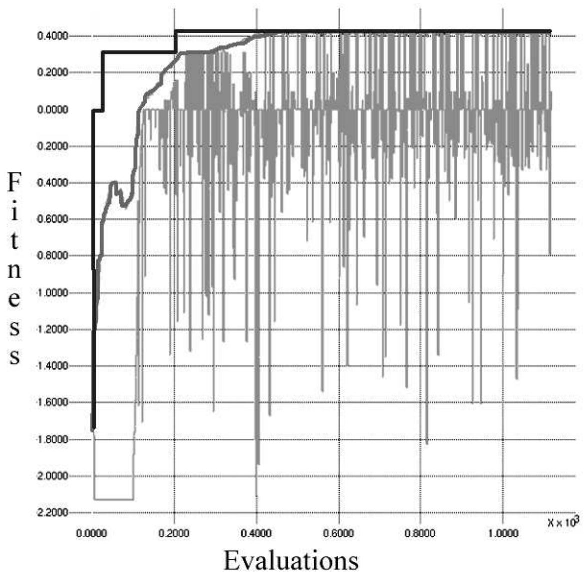


Fig. 6. Min/Max/Avg of Attackers Evolving Against a Hand-coded Defender

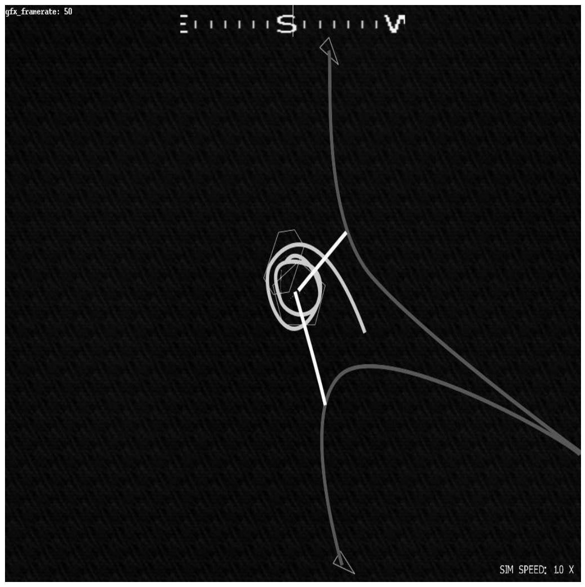


Fig. 7. Behavior Exhibited by Evolved Attacker

strategy, but was very ineffective against any other strategy. If the attackers wait for a short while at the beginning of the mission, the defender will leave the area and then they are then free to attack the oil platform at their leisure.

We found both the offensive and defensive strategies to be fragile, the attackers had no avoidance plan if the defender came after them, and the defender was only effective if the attackers ran off in that direction. This was a problem of over specialization and failed generalization, and to overcome it we turned to co-evolution.

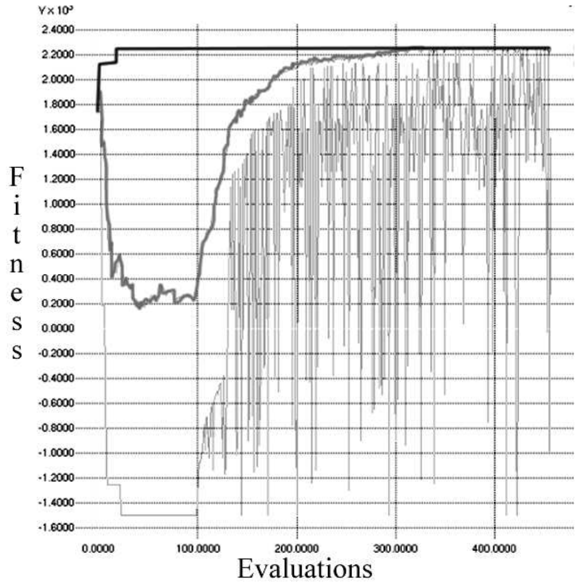


Fig. 8. Min/Max/Avg Fitness of a Defender Evolving against the Evolved Attacker

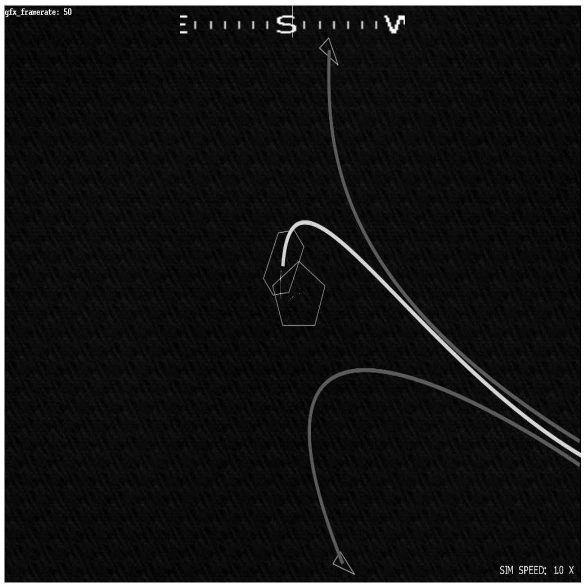


Fig. 9. Behavior Exhibited by Evolved Defender

C. Co-Evolution

Co-evolution occurs when the evaluation of an individual is dependent upon other individuals. We implemented co-evolution with a traditional two population model, with one population containing attacking strategies, and the other containing defending strategies. We evaluated individuals by playing them against un-evaluated individuals in the other population, with fitness calculated as before. The goal being an ”arms race” whereby each side is constantly innovating new strategies in order to better their opponent.

D. Results: Co-evolving Attackers and Defenders

To implement co-evolution we run two genetic algorithms - one evolving attackers, and one evolving defenders. We play unevaluated members from each population against each other, and calculate fitness as before. Again we allowed each GA to evaluate 1000 candidate strategies. Figure 10 shows the minimum, maximum, and average fitness in the two populations over time. Examination of the individuals produced at various stages helped elaborate on the dynamics present within the system. The attackers quickly duplicated their one pass attack as they had done before. Several evaluations later the defender learned to chase them down as it had down before. The attackers then learned to circle around the defender and attack the oil platform again. The defender then learned a good generalized behavior, alternating between patrolling near the oil platform and chasing off the most threatening attackers. The attackers then settled into a hit and run behavior, where they try to get to the far side of the destroyer to attack from as shown in Figure 11. Both attacker and defender learned generalized behaviors, similar to those we had tried to develop in our hand-coded behaviors. The co-evolved versions were generally superior however, with the attackers frequently out-maneuvering the destroyer, and the destroyer vigorously defending the oil platform.

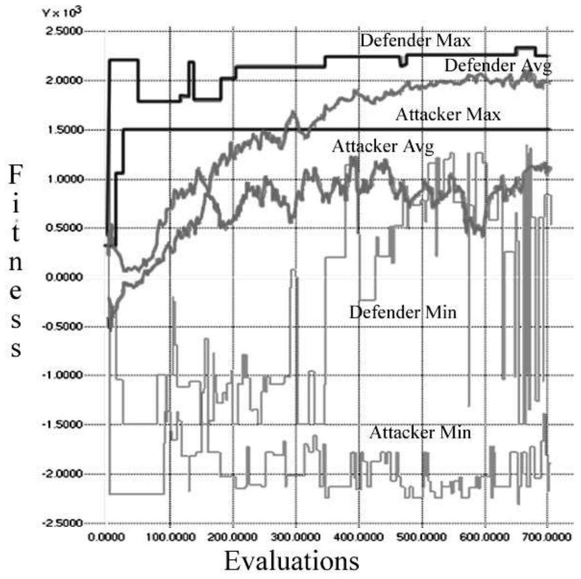


Fig. 10. Min/Max/Avg Fitness’s of Attacker / Defender Co-evolving

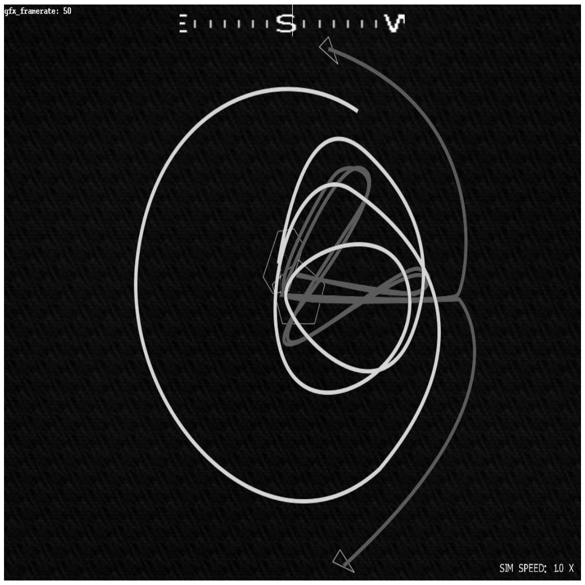


Fig. 11. Behavior of Co-evolved Units

VI. Conclusions and Future Work

Co-evolved influence map trees were capable of producing competent behavior inside our RTS game. While our mission was relatively simple, and the IMTrees were used more as operational controllers than as strategic planners, the IMTrees functioned adequately. The behaviors produced were relatively similar to our hand-coded strategies, but were generally more robust. They were produced without an excess of evaluations, and could be evolved within an hour. Both attacker and defender were of sufficient quality to be used as opponents for humans playing the game. Our results indicate that co-evolving IM Trees is a promising technique, with the potential to evolve strategic players who learn to use complex strategies to win long-term games.

The final plans produced by co-evolution were effective, but not as coordinated as we had originally hoped. The desired behavior was that of a coordinated attack-distract strategy, which we rarely saw the attackers exhibit. This limitation is due to our implementation of the IMTrees, primarily in how they were bound to individual units. Each tree was being used as an individual unit controller. Both trees are identical, differing only in the influence maps relative to the assigned unit, so it is difficult for the attackers to keep from going to the same point. Also since the tree we provided to the GA was relatively small, there was no room for complexity on the level of effective group coordination to emerge. More recent work appears to overcome this problem by using a single influence map tree to represent each side. The IMTree produces a list of objectives and a rough description of which units would be best for them. An allocation GA then allocates individual units and groups to these objectives. This has thus far produced superior results with the attackers effectively coordinating against the defender, and is the primary direction for our current research. The other avenue of future work is that of increasing the complexity present in the mission and the game. An element of stealth has been added to the game, where attackers can hide behind neutral boats in order to approach and hide undetected. Neutral traffic is also being used, requiring the destroyer to maneuver around, and not fire upon, neutral boats while trying to defend a moving ally. Combined with stealth this greatly evens the odds towards the faster attackers. Evolution of the structure of the tree is also a major step under development, allowing strategies to evolve increasing levels of complexity over time, without a steep initial learning curve. Our implementation of co-evolution was also very basic, and future work would include a more complicated system, such as using a hall-of-fame system or a maintaining a sub-sampled population of opponents to test against. Fitness sharing, or some other form of speciation would also be good, particularly to protect and encourage more complicated strategies to develop. These techniques [14] were developed for improving the performance of co-evolution, and would likely lead to faster more consistent improvement. Pareto co-evolution [15] would also provide similar improvements, helping to develop and maintain different attacking and defending strategies within the population.

VII. Acknowledgments

This material is based upon work supported by the Office of Naval Research under contract number N00014-05-1-0709.

References

[1] Angeline, P.J., Pollack, J.B.: Competitive environments evolve better solutions for complex tasks. In: Proceedings of the 5th International Conference on Genetic Algorithms (GA-93). (1993) 264–270

[2] Fogel, D.B.: Blondie24: Playing at the Edge of AI. Morgan Kauffman (2001)

[3] Samuel, A.L.: Some studies in machine learning using the game of checkers. IBM Journal of Research and Development **3** (1959) 210–229

[4] Pollack, J.B., Blair, A.D., Land, M.: Coevolution of a backgammon player. In Langton, C.G., Shimohara, K., eds.: Artificial Life V: Proc. of the Fifth Int. Workshop on the Synthesis and Simulation of Living Systems, Cambridge, MA, The MIT Press (1997) 92–98

[5] Tesauro, G.: Temporal difference learning and td-gammon. Communications of the ACM **38** (1995)

[6] Inc., R.E.: Dawn of war (2005, http://www.dawnofwargame.com)

[7] Studios., B.: Supreme ruler 2010 (2005)

[8] Blizzard: Starcraft (1998, www.blizzard.com/starcraft)

[9] Studios, E.: Age of empires 3 (2005, www.ageofempires3.com)

[10] Louis, S.J., Miles, C., Cole, N., McDonnell, J.: Learning to play like a human: Case injected genetic algorithms for strategic computer gaming. In: Proceedings of the second Workshop on Military and Security Applications of Evolutionary Computation. (2005) 6–12

[11] Gibbons, R.: Game Theory for Applied Economists. Princeton University Press (1992)

[12] Zobrist, A.L.: A model of visual organization for the game of go. In: AFIPS Conf. Proc. (1969) 34, 103–112

[13] Sweetser, P.: Strategic decision-making with neural networks and influence maps. AI Game Programming Wisdom 2 (2001) 439–446

[14] Rosin, C.D., Belew, R.K.: Methods for competitive co-evolution: Finding opponents worth beating. In Eshelman, L., ed.: Proceedings of the Sixth International Conference on Genetic Algorithms, San Francisco, CA, Morgan Kaufmann (1995) 373–380

[15] Bucci, A., Pollack, J.: A mathematical framework for the study of coevolution. In: Foundations of Genetic Algorithms 7. Proceedings of FOGA VII. (2002)