Towards the Co-Evolution of Influence Map Tree Based Strategy Game Players

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. To overcome the knowledge acquisition bottleneck found in using traditional expert systems, scripts, or decision trees we use genetic algorithms to evolve game players. The spatial decision makers in our game players use influence maps as a basic building block from which they construct and evolve trees containing complex game playing strategies. Information from influence map trees is combined with that from an A* pathfinder, and used by another genetic algorithm to solve the allocation problems present within many game decisions. As a first step towards evolving strategic players we develop this system in the context of a tactical game. Results show the co-evolution of coordinated attacking and defending strategies superior to their hand-coded counterparts.**

Fig. 1. Earth 2160 - Reality Pump Studios

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, popular computer games like Starcraft and Counter-Strike are very different and have not received much attention from researchers [1], [2], [3], [4], [5]. 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. To develop truly competitive opponents these computer players often cheat, changing the nature of the game in their favor, in order to defeat their human opponents [6]. 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 that 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, Earth 2160 (Figure 1), or Age of Empires [7], [8], [9], [10], [11]. 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 players' assets.

Games are fundamentally about making decisions and exercising skills. In contrast to some game genres, RTS games concentrate player involvement primarily around making decisions, the alternative being a game such as a racing games which requires 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.

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

Our 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, and has evolved influence map trees to make spatial reasoning decisions within RTS games [12], [13]. Our players combine these two systems, using genetic algorithms for allocation decisions and influence map trees for spatial reasoning. The spatial decision making system looks at the game world and decides to build a base here, to put a wall up there, and to send a feigning attack over there. An A* pathfinder looks at the feasibility of reaching those objectives, noting that putting up a wall there would be great if there wasn't an enemy army in the way [14]. The allocation system allocates available resources to objectives, deciding that this unit group has the weaponry and is in position to lay siege to the city. These systems combine into a game player, which is capable of carrying out coordinated strategies.

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 our game playing 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 [15]. We then develop a fitness function which evaluates these decision makers based upon their in-game performance. 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 and test our architecture within the context of a 3D computer RTS game. Our architecture ties together a spatial reasoning system based on influence map trees, with a genetic algorithm performing allocations. Encoded as individuals of a genetic algorithms population, these players are evolved to improve their game-playing abilities. Results show this is effective, with players showing coordinated game-playing strategies. We describe the spatial decision making system, and how it ties into the path-finding and genetic algorithm allocation systems. We then detail 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 an analysis of the system's performance, including the behaviors produced by evolution. Finally we discuss directions for the continuation of this research.

II. Representation - Game Player

Each individual in the population represents a game-playing strategy. RTS games are primarily about making spatial reasoning and resource allocation decisions. We first use a combination of influence maps to do spatial reasoning, and later use genetic algorithms to solve the allocation problems. An objective zoner converts the influence maps into objectives for player units to carry out. Each objective is a task to be carried out at some point in space: attack here, defend this, or move here. Meta-data is attached to each objective, describing what kinds of units to allocatd to them. For example a "siege enemy city" objective requests long range artillery, while a feigning attack objective requests fast and disposable troops. A genetic algorithm then allocates unit groups to these objectives, using the information available to solve the underlying allocation problems. An A* pathfinder is used to determines the spatial costs involved in these allocations: objectives that are far away are more costly, as are objectives which require traversing dangerous territory. This final allocation takes into account how beneficial each objective is perceived to be, how well the unit composition of the groups match the units requested by the objective, and how readily those unit groups can reach those objectives. The overall architecture is shown in Figure 2. Our game players represent their spatial reasoning strategy within influence maps, we describe these influence maps in the next section.

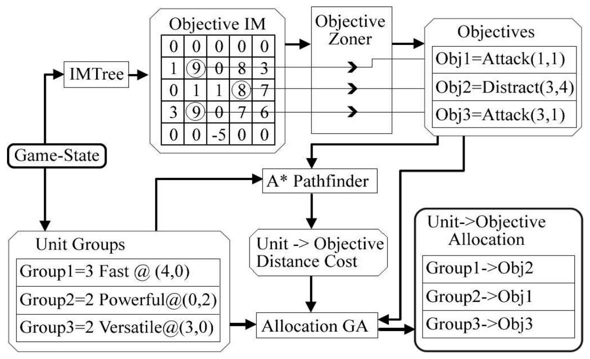
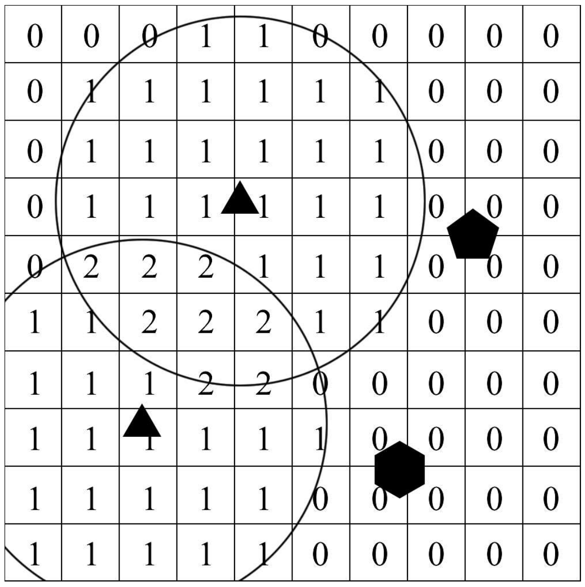


Fig. 2. Game Player Architecture

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. Influence maps evolved out of work done on spatial reasoning within the game of Go and have been used sporadically since then in games such as Age of Empires [16], [11]. Influence maps combine together to form spatial decision making strategies. 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 3 is a visualization of an influence map, with the triangles in the game world increasing the values of squares within some radius of their location.

We create and combine Several IM's to form our 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



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. 3. An Influence Map

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 next step is to analyze the resultant IM and translate it into orders which can be assigned to units. We are looking for multiple points to assign to multiple unit groups, so we use the system described in Section V.

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 evolved a neural network which took every square from every IM as an input, and produced the squares of the final IM as output[17] . 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

We combine IM's within a tree structure instead of the traditional list [16]. Each tree represents a complete decision making strategy, and is encoded within an individual in 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 [16]. 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 4 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 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.

A. The Mission

To test our players we created the mission shown in Figure 5. Two small cigarette boats - triangles at top, attempt to attack an oil platform - pentagon, which is being guarded by a destroyer - hexagon. The cigarette boats are fast, maneuverable, and equipped with rocket propelled grenade launchers. Their primary advantage over the defending destroyer is that there are two of them and that they can quickly accelerate, decelerate and turn. The destroyer on the other hand is also quite fast, with a higher top speed than 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. While the cigarette boats have slightly more range, the destroyer has far more firepower.

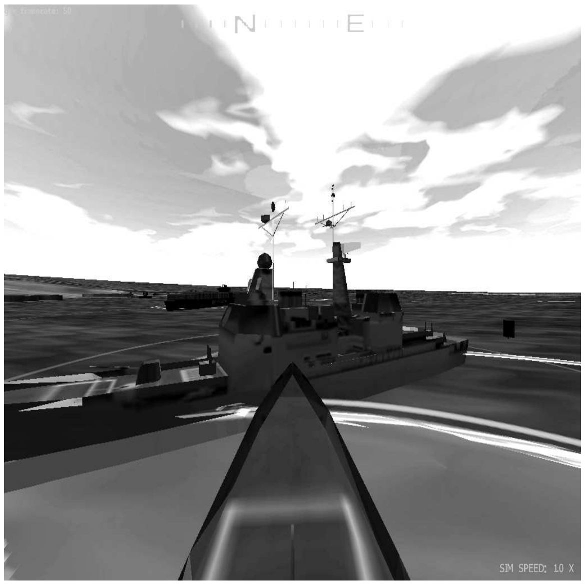


Fig. 4. Lagoon

This mission was chosen as it was relatively simple, and it required the players to understand the effectiveness of their units, with the possibility of evolving coordinated attacks. 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.

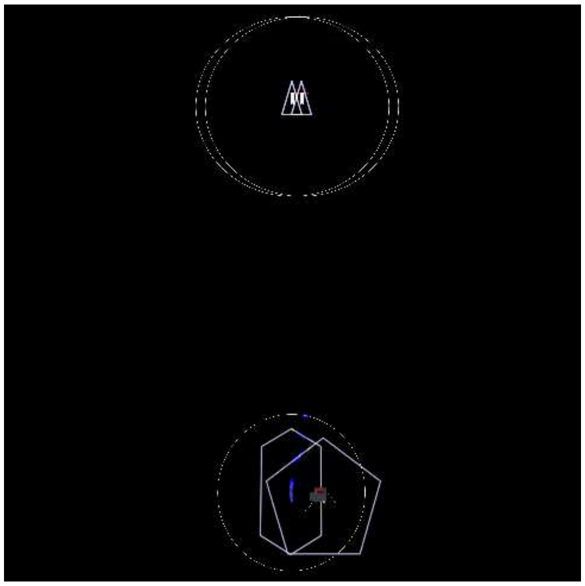


Fig. 5. Mission

IV. Player Implementation

Both sides of the game have a player controlling them, with each player having its own influence map tree and allocation GA. The players do continuous processing throughout the game, calculating a small amount of an influence map, or running another evaluation in the allocation GA at each slice of time. In this way the players do complex calculations without consuming too much processing power. Currently it takes between 10-20 seconds for each player to complete a processing cycle, at which point new objectives are assigned to player units. We next describe our implementation of the various player components, starting with the influence maps and their combination into a tree.

A. Influence Map Tree Implementation

The IM’s calculate the value of their squares with IM functions based on which units are near those squares as shown in Figure 3. 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 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 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 3 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].

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.

V. Objective Zoner

Processing the influence map produced by the influence map tree into objectives is the job of the objective zoner. An objective is a task for a unit group to carry out at some point in space. The objective zoner should reduce the influence map to a set of objectives representing its most important points, including all the distinct peaks in the landscape without being redundant. In Figure 2 the IMTree produces as output the objective IM, from which the objectives are parsed. The zoner in response picks three key points, the two peaks on the left, and one of the points from the plateau on the right. The zoner determines the first two points correspond to attack actions, because the influence map which produced them was linked to that behavior, while the third point and its corresponding IM was linked to a move behavior. This gives three objectives, attack those two points, or move to this point and distract the enemy.

Our objective zoner uses a simple algorithm to create the objectives. It finds the highest point in the influence map that is not to close to an already chosen objective, and takes it as an objective. It then repeats the process, until no eligible points are left. Points are ineligible if their value is less then some ratio of the highest point, $1/2$ in this case. They are also ineligible if they are too close to a previously assigned objective. Currently all objectives carry a generic attack-move behavior, where they move directly to the target point firing at enemies that come within range.

VI. A* Pathfinder

Once the objectives have been calculated, an A* router determines how accessible they are. This is done by looking at the unit groups available to the player, and searching for how easily they can reach those objectives. This information is used by the allocation genetic algorithm to keep units from being allocated to objectives they cannot reach, and prioritizes more accessible objectives.

We used a traditional A* pathfinder to do path-finding, A* is shown to always produce the optimal path through a graph so long as it has a proper underestimate, which we have. Our pathfinder searches through squares in an influence map to find the optimal path from one point to another. The A* influence map is separate from the players influence map tree, with each square representing how preferable it is to route through. It has high negative values near powerful enemies, and more positive numbers in open water. Because the A* router uses an influence map, the search space could be arbitrarily complicated, routing units so that they try to stay close to neutral units, or as far from land as possible.

VII. Allocation GA

AllocGA, a genetic algorithm, allocates units to objectives. AllocGA is a non-generational GA which uses single point crossover, bitwise mutation and roulette wheel selection. It determines its resources currently available, and maps them to appropriate objectives taking into account the information provided by the other systems. Each individual in AllocGA's population encodes an allocation table, listing which objective each unit is assigned too. The fitness for each individual is a summation of the benefit expected from allocating each unit group to its objective. The benefit for allocating a group to an objective is given by: 1) the expected benefit from the objective 2) how well that groups units match the units requested for the objective 3) how easily the unit group can reach that objective. The expected benefit for an objective is the value from its point in the influence map. How well units match those requested is based upon the composition of the group, and the meta-data assigned to those objectives as discussed in SectionVII-.1. The penalties for risk incurred, and travel distance is the total cost associated with the route from unit to objective found by the A* pathfinder. Attacking an enemy city might yield tremendous benefit, but not for a group of units without the appropriate weaponry, or for units occupied on the far side of the map.

1) Objective Meta-Data: To determine which units to allocate to which objectives we attach meta-data describing what kind of units would be useful to each objective. We use three coefficients, representing the power, speed, and value of the units allocated. To each unit we associate three abstract values, a power rating summarizing how effective it is in combat, a speed rating summarizing how fast and maneuverable it is, and a value rating representing its cost. Each is an abstract hand set value summarizing the complex workings of the entity, the destroyer's has a high power, the cigarette boats have high speed ratings, and the oil-platform has a high value rating. AllocGA calculates benefit based how the units allocated to an objective match up with the coefficients attached to the objective. If an objective has a high power coefficient, then allocating powerful units increases the fitness of that allocation. Conversely if an objective has a negative value coefficient, then allocating expensive units reduces the fitness. Each units attributes are multiplied by the corresponding coefficient, and the summation is applied to the fitness of that allocation. By changing these values an influence map can be suited for fast cheap units, powerful expensive ones, or weak valuable units (maybe a hide in the rear IM). Each objective can also have a unit cap, beyond which no benefit is received for adding units, so that distracting with 10 units is not more beneficial than using 1.

VIII. Hand-Coded Players

To test this system we first develop hand-coded players for both sides, tuning their behavior over a few games to test how well the players work. Our hand-coded attacker work by using an OR node on two child subtrees. The first subtree represents an attack behavior which takes the weighted sum of two nodes. The first node has high values near vulnerable enemies, the second has large negative values near powerful enemies. This gives points near vulnerable enemies, but away from powerful ones The second subtree represents a distract behavior where the cigarette boat tries to stay just out of range of the destroyer, baiting it into following it and in the process abandoning the oil platform. The distract child has two children of its own, the first representing a ring of points just outside the destroyers range, and the second with high values away from the oil platform To generate the ring of points outside the destroyers range it sums two influence maps - one with radius equal to the destroyers weapon range but with negative points and one with a slightly larger radius and positive points. This gives a ring of positive points just outside of weapons range. The second child of the distract behavior represents points away from the oil platform, and by multiplying this with the ring outside of weapons range we get points just outside of the destroyers weapons range that are away from the oil-platform.

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 hand-coded IM trees worked reasonably well, with the attackers trying to out-maneuver the defender as it patrols around the oil-platform.

We found our hand-coded attackers showed a reasonable level of coordination, with one boat distracting while the other attacked. The defender was effective, staying near the oil platform until the cigarette boats approached. If they approached it would try to chase them off, firing if they got too close. There behavior was quirky however, and not particularly well rounded. If the defender did not take the bait it could often catch the other boat by supervise, also the attackers were not efficient with their firepower, as the later evolved behaviors showed that by switching places the attackers could do much more damage. They also had a hard time getting from one side of the destroyer, often entering its field of fire and being destroyed. The defender was very often fooled by the attackers as well, lured away from the oil-platform it was trying to protect. To improve upon both of these behaviors we turned to evolutionary techniques, allowing the GA to evolve IMTrees for controlling units in our game.

IX. Evolving Players

We evolved our players with a non-generational genetic algorithm (PlayerGA) with roulette wheel selection, one point crossover and bitwise mutation. The influence map trees used by the players are encoded within individuals of PlayerGA. Crossover took place with 75% probability, and the bitwise mutation probability was chosen to give on average 2 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. PlayerGA 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.

X. Results

Our hand-coded attacker had a basic attack-distract behavior, with one cigarette boat trying to distract and occupy the destroyer while the other went for the oil-platform. Our basic defender spent most of its effort chasing after the attackers, hoping to cut them off and broadside them with its machine guns. Our hand-coded attackers were reasonably effective - winning most of the missions, but often making mistakes. The attackers would easily occupy the destroyer while it chased them, but if it switched who it was chasing they would often try to cut across its field of fire. To improve this we first evolved an attacker against our hand-coded defender. This gave better behavior, with the evolved attackers being more flexible and reliable. We then evolved the defender, with the defender becoming a bit more robust in how it would deal with two attackers, not getting lured as easily away from the oil platform. Finally we evolve the two populations simultaneously, seeing more types of behaviors from both players, before ultimately producing two well rounded players.

A. Results: Evolving the Attacker

The GA first evolved our attacker, evaluating 1000 individuals against our hand-coded defender. While we ran the system multiple times, we will discuss a single representative run which illustrates the results we consistently achieved. The attackers eventually discover a reasonably good attack-avoid strategy, staying well away from the destroyer while trying to get close to the oil platform. Over the following evaluations this evolves into an attack-distract strategy, where the attacks split their time occupying the destroyer and attacking the oil-platform. Unlike our hand-coded attacker they were not reluctant to switch roles, with one boat distracting for several seconds then going back to the oil platform. This allows them to attack the platform, and spend their long reloading time distracting the destroyer, limiting the amount of shots they fired on the destroyer - who is more heavily armored. They were also much more cautious about approaching the destroyer, going well out of their way to avoid it. This avoiding the problem of our hand-coded attacker, whereby it would occasionally skim the destroyers firing range, taking heavy fire. The evolved attacker proved frustrating to play against as an opponent, as it was very chaotic in its actions. While psychologically effective it did make mistakes, but overall it represented a significant improvement from our hand-coded attacker.

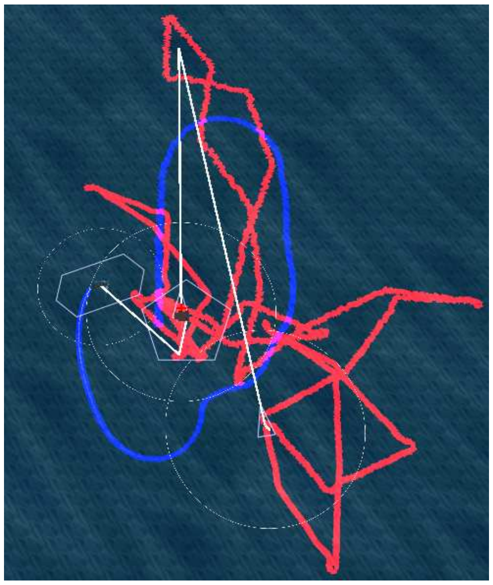


Fig. 6. Behavior Exhibited by Evolved Attacker

B. Results: Evolving the Defender

The evolved attackers were effective against the hand-coded defender, coordinating an effective attack-distract strategy. We next re-ran the genetic algorithm to evolve the defender to see if it could find a counter to the attackers strategy. The attack distract behavior capitalizes well on the advantage given to the attackers, making it difficult for the destroyer to effectively defend. The defenders evolved did surpass the quality of our own hand-coded defenders however, learning how to trick the attackers into making mistakes. Figure 7 shows an exceptional defense, where the defender pushes both attackers back by manipulating their constant switches in roles. While the evolved attacker and evolved defender were effective, particularly against each other, they made obvious mistakes against human opponents. To improve our players further we utilized co-evolution, aiming to generate ever more robust players.

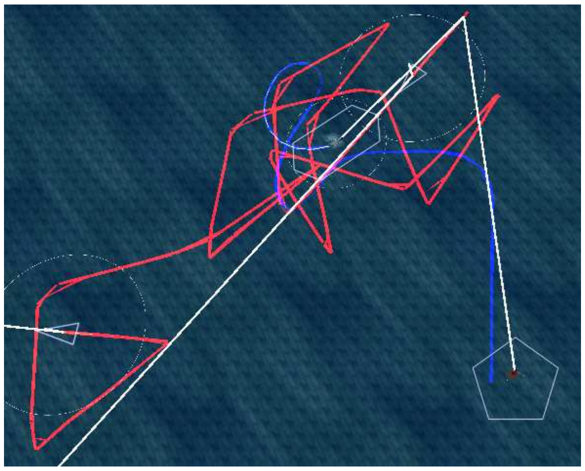


Fig. 7. Behavior Exhibited by Evolved Defender

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. As before we allowed each GA to evaluate 1000 candidate strategies. Figure 8 shows the minimum, maximum, and average fitness in the two populations over time. At first there is chaos, with both players using random strategies. Effective attackers and defenders start to emerge however, with the attacker learning to go for the oil platform and the defender learning to go for the attackers. The attackers suffer for a few hundred generations, trying to learn an attack-distract or an attack-avoid behavior. Eventually those start to emerge, and their fitness rises dramatically. This leads to improvements in the defenders AI, learning not to be lured away from the oil platform, and to keep its speed up. Ultimately they develop the behaviors shown in Figure 9 - the attacker develops a well rounded attack-distract-avoid behavior, and the defender develops a diligent defensive behavior. The attackers spends less time distracting then before, preferring to stay on the opposite side of the oil-platform and fight. One boat will occasionally lure the defender away, and then return while the defender turns around. The attackers also tend to stay far away, generally opposite sides as shown in Figure 9, which makes them much more flexible than if they bunch up. The defenders behavior was very protective, generally staying very close to the oil-platform. It was difficult to lure off, and did a good job of overcoming its slow turning rate by staying in good positions.. Both attacker and defender learned generalized behaviors, similar to those we had tried to develop in our hand-coded behaviors. The co-evolved players were superior to the hand-coded players, with the attackers clearly out-maneuvering the destroyer, and the defender doing its best to defend the oil-platform. Co-evolution gave them the robustness necessary to play against human opponents.

XI. 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. By combining influence maps with genetic algorithms we produced behavior that was much more coordinated than in either of the previous systems. The attackers learned how to effectively work as a

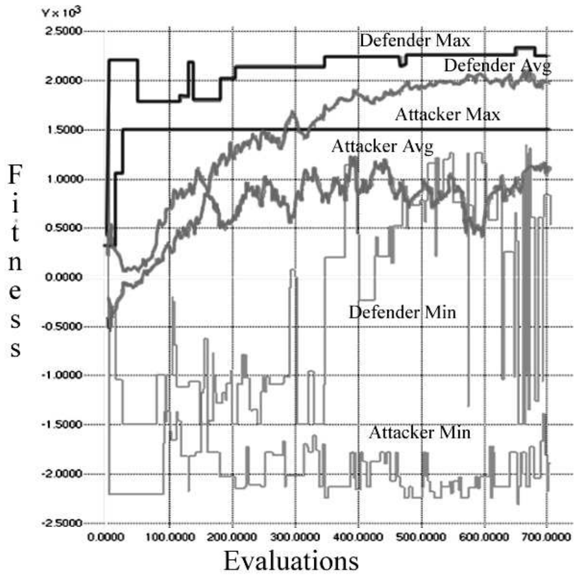


Fig. 8. Min/Max/Avg Fitness's of Attacker / Defender Co-evolving

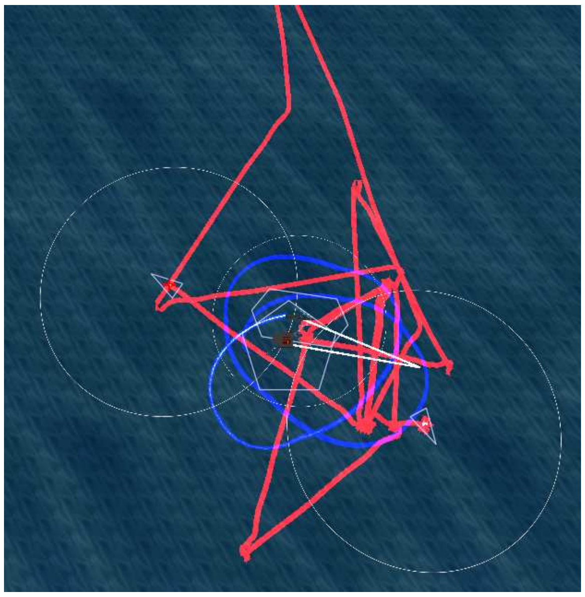


Fig. 9. Final Co-Evolved Players

team, taking advantage of their unit's abilities to overcome a more powerful defender. This is in contrast to previous work, where the attackers functioned independently and were defeated by the defender [12]. The final behaviors were robust and effective, both against other evolved players and against human opponents. 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 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 complicates the games for both players. 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. Future work would expand our implementation of co-evolution, utilizing 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, protecting and encouraging more complicated strategies to develop. These techniques were developed for improving the performance of co-evolution, and would likely lead to faster more consistent improvement [18]. Pareto co-evolution would also provide similar improvements, helping to develop and maintain different attacking and defending strategies within the population [19].

XII. Acknowledgments

This material is based upon work supported by the Office of Naval Research under contract number N00014-03-1-0104.

References

[1] P. J. Angeline and J. B. Pollack, "Competitive environments evolve better solutions for complex tasks," in *Proceedings of the 5th International Conference on Genetic Algorithms (GA-93)*, 1993, pp. 264–270. [Online]. Available: citeseer.ist.psu.edu/angeline93competitive.html
[2] D. B. Fogel, *Blondie24: Playing at the Edge of AI*. Morgan Kauffman, 2001.
[3] A. L. Samuel, "Some studies in machine learning using the game of checkers," *IBM Journal of Research and Development*, vol. 3, pp. 210–229, 1959.
[4] J. B. Pollack, A. D. Blair, and M. Land, "Coevolution of a backgammon player," in *Artificial Life V: Proc. of the Fifth Int. Workshop on the Synthesis and Simulation of Living Systems*, C. G. Langton and K. Shimohara, Eds. Cambridge, MA: The MIT Press, 1997, pp. 92–98.
[5] G. Tesauro, "Temporal difference learning and td-gammon," *Communications of the ACM*, vol. 38, no. 3, 1995.
[6] J. E. Laird and M. van Lent, "Human-level ai's killer application: Interactive computer games," 2000. [Online]. Available: http://ai.eecs.umich.edu/people/laird/papers/AAAI-00.pdf
[7] R. E. Inc., "Dawn of war," 2005, http://www.dawnofwargame.com.
[8] B. Studios., "Supreme ruler 2010," 2005.
[9] R. Pump., "Earth 2160," 2005.
[10] Blizzard, "Starcraft," 1998, www.blizzard.com/starcraft. [Online]. Available: www.blizzard.com/starcraft
[11] E. Studios, "Age of empires 3," 2005, www.ageofempires3.com. [Online]. Available: www.ageofempires3.com
[12] C. Miles and S. J. Louis, "Co-evolving real-time strategy game playing influence map trees with genetic algorithms," in *Proceedings of the International Congress on Evolutionary Computation, Portland, Oregon*. IEEE Press, 2006, pp. 0–999 999 999 999.
[13] S. J. Louis, C. Miles, N. Cole, and J. McDonnell, "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, pp. 6–12.
[14] B. Stout, "The basics of a* for path planning," in *Game Programming Gems*. Charles River media, 2000, pp. 254–262.
[15] R. Gibbons, *Game Theory for Applied Economists*. Princeton University Press, 1992.
[16] A. L. Zobrist, "A model of visual organization for the game of go," in *AFIPS Conf. Proc.*, 1969, pp. 34, 103–112.
[17] P. Sweetser, "Strategic decision-making with neural networks and influence maps," *AI Game Programming Wisdom 2*, pp. 439–446, 2001.

[18] C. D. Rosin and R. K. Belew, "Methods for competitive co-evolution: Finding opponents worth beating," in *Proceedings of the Sixth International Conference on Genetic Algorithms*, L. Eshelman, Ed. San Francisco, CA: Morgan Kaufmann, 1995, pp. 373–380.

[19] A. Bucci and J. Pollack, "A mathematical framework for the study of coevolution," in *Foundations of Genetic Algorithms 7. Proceedings of FOGA VII*, 2002.