Learning to Play Like a Human: Case Injected Genetic Algorithms Applied to Strategic Computer Game Playing

Chris Miles Sushil J. Louis Nicholas Cole
Evolutionary Computing Systems Lab
Department of Computer Science
University of Nevada
Reno - 89557
{miles,sushil,ncole}@cs.unr.edu

John McDonnell
SPAWAR Systems Center
San Diego, CA
john.mcdonnell@navy.mil

***Abstract*— We use case injected genetic algorithms to learn how to competently play computer strategy games. Strategic computer games involve long range planning across complex dynamics and imperfect knowledge presented to players requires them to anticipate opponent moves and adapt their strategies accordingly. In this paper, we address the problem of acquiring knowledge learned from human players, in particular we learn general routing information from a human player in the context of a strike planning game. By incorporating case injection into a genetic algorithm, we show methods for learning general knowledge from human players to incorporate into future plans. Results show that with an appropriate representation, case injection is effective at biasing the genetic algorithm toward producing plans that contain important strategic elements used by human players.**

I. Introduction

We use case injected genetic algorithms to learn to play strategic games [1]. In particular we attack the problem of using case injection to bias a genetic algorithm player such that it better emulates the playing styles of human players it has seen in the past. Our research focuses on a strike force asset allocation game which maps to a broad category of resource allocation problems in industry and the military. Genetic algorithms can be used in our game to robustly search for effective strategies. These strategies may approach game optimal strategies but they do not necessarily approach real world optima as the game is an imperfect reflection of reality. Humans with past experience playing the real world game tend to include external knowledge when producing strategies for the simulated game. Incorporating knowledge from the way these humans play should allow us to carry over some of this external knowledge. Our results show that case injection combined with a flexible representation can bias the genetic algorithm towards producing strategies similar to those learned from human players. Beyond playing similarly in a particular mission, the genetic algorithm can use strategic knowledge across a range of similar missions to continue to play as the human would.

We seek to produce a genetic algorithm player (GAP) that can play on a strategic level and learn to emulate aspects of strategies used by human players. Our goals in learning to play like humans are:

Fig. 1. Game Screen-shot

1) To make GAP a more interesting opponent. Humans generally enjoy playing against opponents they can relate to and personify.
2) GAP should be able to function as a trainer, a player who plays not just to win but to teach their opponent how to better play the game, in particular to prepare them for future play against human opponents. This would allow us to use GAP for acquiring knowledge from human experts and transferring that knowledge to amateur human players without the expense of individual training with experts.
3) We want to use GAP for decision support, whereby GAP provides suggestions and alternative strategies to humans

actively playing the game. Strategies more compatible with those being considered by the humans should be more likely to have a positive effect on the decision making process.

4) We would like to incorporate dynamics into the game without the expense of simulating them. Consider a game involving armies doing battle, both players rotate out their front line troops to let them rest. Whether or not the game simulation has an accurate model of fatigue the result is much the same, playing in anticipation of this dynamic (fatigue) is equivalent to implementing it. If GAP can incorporate game dynamics from watching humans play, it should deepen the feeling of strategy involved in the game.

These roles require GAP to play with objectives in mind besides that of winning – these objectives would be difficult to quantify inside the evaluator. As humans can function effectively in these regards, learning from them should help GAP better fulfill these responsibilities.

We attack the problem of modeling humans through case-injected genetic algorithms. The genetic algorithm searches for an optimal strategy while case injection biases it to contain elements from strategies used by humans in the past, how it does this is explained in the next paragraph. Used in conjunction we hope to produce near optimal strategies that incorporate important information external to the evaluation function. We work on a strategic game, specifically, strike force asset allocation which consists primarily of allocating a collection of strike assets to a set of targets. We have implemented this game on top of a professional game engine making it more interesting than a pure optimization problem. The game involves two sides: Blue and Red, Blue allocates a set of platforms (aircraft) to attack Red's targets (buildings). Red has defensive installations (threats) that complicate Blue's planning, as does the varying effectiveness of Blue's weapons against each target. Potential new threats and targets can also "pop-up" on Red's command in the middle of a mission, requiring Blue to be able to respond to changing game dynamics. Both players seek to minimize the damage they receive while maximizing the damage dealt to their opponent. Red plays by organizing defenses in order to best protect its targets. Red's ability to play popups can also affect its strategy. For example, feigning vulnerability can lure Blue into a pop-up trap, or keep Blue from exploiting a weakness out of fear of such a trap. Blue plays by allocating its platforms and the assets (weapons) they carry as efficiently as possible in order to destroy the targets while minimizing risk. Risk is determined by many factors, including the platform's route, the effect of accompanying wingmen, and the presence of threats around chosen targets. GAP develops strategies for the attacking strike force, including flight plans and weapon targeting for all available aircraft. When confronted with popups, GAP responds by replanning with the genetic algorithm to produce a new plan of action. Beyond producing near optimal strategies we would like to bias it towards producing solutions similar to those it has seen used by humans playing Blue in the past. We do this so that GAP can be a better and more versatile player by learning strategies from human experts. Specifically we want to show that when using case injection the genetic algorithm more frequently produces plans similar to those a human has played in the past on the same mission. When that information from the human is used by GAP playing a different mission, we show that GAP continues to play strategies closer to the style of the human than GAP's non injected counterpart.

Case-injected genetic algorithms work by saving individuals from the population of a GA, and later introducing them into a GA solving a similar but different problem. Louis showed that case injection improves convergence speed and the quality of solutions found by biasing the current search toward promising regions identified from experience [1], [2]. In this paper, the system *acquires cases from humans* for injection into GAP's population. The idea is to automatically acquire cases by instrumenting the game interface to record all human decision making during game play. Our goal is not only to improve the GA's performance, but to bias the search using knowledge that is external to the actual evaluation and fitness of an individual plan – knowledge being expressed by expert humans in their formation of game plans.

Previous work in strike force asset allocation has been done in optimizing the allocation of assets to targets, the majority of it focusing on static pre-mission planning. Griggs [3] formulated a mixed-integer problem (MIP) to allocate platforms and assets for each objective. The MIP is augmented with a decision tree that determines the best plan based upon weather data. Li [4] converts a nonlinear programming formulation into a MIP problem. Yost [5] provides a survey of the work that has been conducted to address the optimization of strike allocation assets. Louis [6] applied case injected genetic algorithms to strike force asset allocation, showing results consistent with the effectiveness of our GA.

A large body of work exists in which evolutionary methods have been applied to games [7], [8], [9], [10], [11]. However the majority of this work has been applied to board, card, and other well defined games. Such games have many differences from popular real time strategy (RTS) games such as Starcraft, Total Annihilation, and Homeworld[12], [13], [14]. Many traditional (board, card, paper) games use entities (pieces) that have a limited space of positions (such as on a board) and restricted sets of actions (well defined movement). Players in these games also have well defined roles and the domain of knowledge available to each player is clearly identified. These characteristics make the game state easier to specify and analyze. In contrast, entities in our game exist and interact over time in continuous three dimensional space. Entities are not directly controlled by players but instead sets of algorithms control them in order to meet goals outlined by players. This adds a level of abstraction not found in those traditional games. In most of these computer games, players have incomplete knowledge of the game state, and even this domain of each a player's knowledge is difficult to identify. John Laird [15], [16], [17] surveys the state of research in using Artificial

Intelligence (AI) techniques in interactive computers games. He describes the importance of such research and provides a taxonomy of games. Several military simulations share some of our game's properties [18], [19], [20], these however are military simulations while ours is not intended to perfectly model the dynamics of real situations but to provide a platform for research in strategic planning and to have fun.

In this paper we first define the problem being attacked, including the mission being played. Then we outline how we use a GA to play the game, including delving into the architecture and how case injection works. Next we discuss how case injection is used to acquire and use knowledge from human players. Finally we show results that the GA can play the game, and by using case injection we can significantly increase its likelihood of playing like a human.

II. The Mission

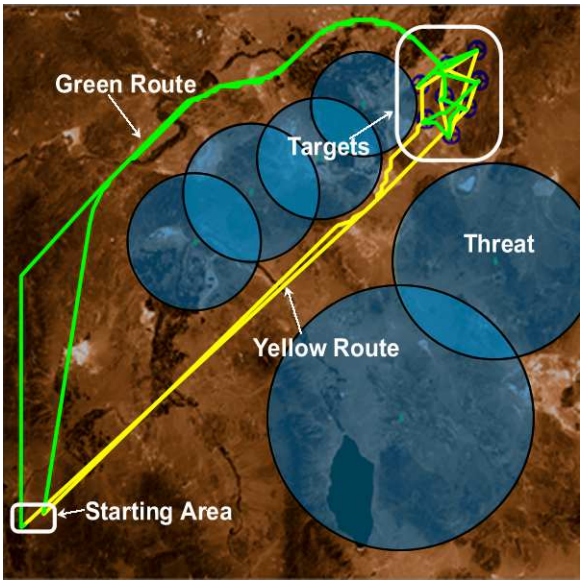

Fig. 2. The Mission

The mission being played is shown in Figure 2. This mission was chosen to be simple, to have easily analyzable results, and to allow the GA to learn external knowledge from the human. As many games show similar dynamics, this mission is a good arena for examining the general effectiveness of using case injection for learning from humans. The mission takes place in Northern Nevada and California, Lake Tahoe is visible near the bottom of the map. Blue possesses one platform which is armed with 8 assets (weapons) and the platform takes off from and returns to the lower left hand corner of the map. Red possesses eight targets distributed in the top right region of the map, and six threats that defend them. The first stage in Blue's planning is determining the allocation of the eight assets. Each asset can be allocated to any of the eight targets, giving $8^8 = 2^{24}$ allocations. The second stage in Blue's planning involves finding routes for each of the platforms to follow during their mission. These routes should be short and simple but still minimize exposure to risk. We categorize Blue's possible routes into two categories. Yellow routes fly through the corridor between the threats, while green routes fly around. The evaluator has no direct knowledge of potential danger presented to platforms inside the corridor area. Because of this, the evaluator optimal solution is the yellow route, since it is the shortest. The human expert however, understands the potential for danger as the corridor provides the greatest potential for a pop-up trap. Knowing this the green route is the human optimal solution. Our goal is now to bias the GA to produce the green route, while still optimizing the allocation. Teaching GAP to learn from the human and produce green strategies even though yellow strategies have higher fitness is the goal of this research. Our way of measuring the state of entities in the game is probabilistic and we describe it in the next section.

A. Probabilistic Health Metrics

In many games, entities posses hit-points which represents their ability to take damage. Each attack then removes a number of hit-points and when reduced to zero (0) hit-points that entity is destroyed. In reality weapons have a more hit or miss effect, whereby they entirely destroy things or leave them functional. A single attack may destroy an entity or multiple attacks may have no effect. This paradigm introduces a high amount of stochastic error into the game. Evaluating a plan can result in outcomes ranging from total failure to perfect success, which makes it difficult to compare two plans. By taking a statistical analysis we achieve better results. Consider the state of each entity at the end of the mission as a random variable. Identifying the expected values for those variables becomes one means to judge the effectiveness of a plan. These expected values can be estimated by playing a number of games for each plan and averaging the results. However doing multiple runs to determine a single evaluation increases the computational expense many-fold.

We use a different approach based on probabilistic health metrics. Instead of monitoring whether or not an object has been destroyed we monitor the probability of its survival up until that point in time. Being attacked no longer destroys objects and removes them from the game, it reduces their probability of survival from then on according to Equation 1.

$$S(E) = S_{t_0}(E) * (1 - D(E)) \quad (1)$$

E is the entity being considered, which is a platform or target under attack. $S(E)$ represents the chance of that entity surviving past this point in time. $S_{t_0}(E)$ is chance of survival up until the attack. $D(E)$ is the chance of that platform being destroyed by the attack as given by equation 2.

$$D(E) = S(A) * E(W)) \quad (2)$$

$D(E)$ is the chance of destruction by this attack. $S(A)$ is the attackers chance of survival up until the time of the attack. $E(W)$ is the effectiveness of the attackers weapon as given by the weapon-target effectiveness table. This method gives us the expected values of survival for all entities in the game

within one run of the game, thereby producing a representative and non-stochastic evaluation of the value of a plan. As a side effect, we also gain a smoother gradient for the GA to search as well as consistently reproducible evaluations.

III. System Architecture

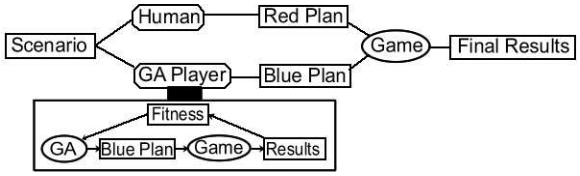

Fig. 3. System Architecture.

Figure 3 shows our system's architecture. The two players, human and GAP, are presented with the mission and given time to prepare their strategy. GAP works by applying its GA to the mission. The GA creates populations of bit strings, which are converted into plans and evaluated. Based on this evaluated fitness individuals are recombined and new plans are produced. We combine a steady state population model, roulette wheel selection, two point crossover and bitwise mutation to form our GA. Production of a full flight plan is complicated, so we next detail the steps involved.

A. Routing

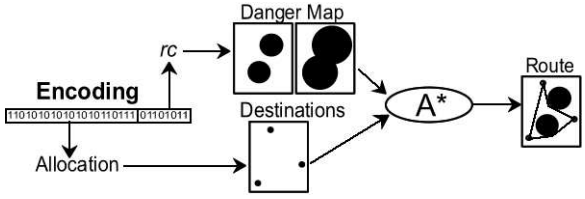

Fig. 4. How Routes are Built From an Encoding.

GAP must be able to produce routing data for Blue's platforms in order to play the game. Figure 4 shows how the A* algorithm is used to build routes [21]. From the allocation of assets to targets we produce an ordered list of waypoints for each platform to visit. In order for platform P1 to use asset A1 on target T3, it has to fly to T3's location. Applying this to all of P1's assets produces a list of waypoints for P1 to visit during its mission. Future work would include ordering information into the chromosome, currently, targets are visited according to the ordering of their respective assets. From the list of waypoints we use a path-finding algorithm to produce more intelligent routes. A number of path-finding algorithms exist, A* was chosen as it is very widely used and comprises the vast majority of path-finding algorithms in games.

Path-finding between destinations is accomplished in two stages. First, the system discretizes the world into a voxel grid. A voxel is a three dimensional cube, with each voxel having a value representing the danger presented to platforms inside it. The voxels are then formed into a graph, where edges connect adjacent voxels in 3D space. By starting A* at the voxel containing the start location, and searching out the most promising neighbor voxels we can explore the graph until we locate a route to the voxel containing the destination. A* is guaranteed to always find the shortest route if it is given a proper underestimate of distance to the goal. This route is a list of waypoints to fly to in order to reach the destination and in two dimensions it is equivalent to a list of street intersections to drive through in order to get from A to B. Due to the large number of waypoints, the routes produced are too cumbersome to use efficiently. Because platforms are not limited to moving through space orthogonally we can remove the majority of waypoints from the route. Imagine a rubber band stretched around a set of pegs placed along a grid, removing unnecessary pegs from the graph leaves pegs outlining threats to avoid, and pegs marking places to visit. Specifically the algorithm works by starting at the beginning of the route, and removing subsequent pegs (waypoints) so long as they do not increase the total risk involved in the route. Once a node has been found that cannot be removed the algorithm repeats from that location.

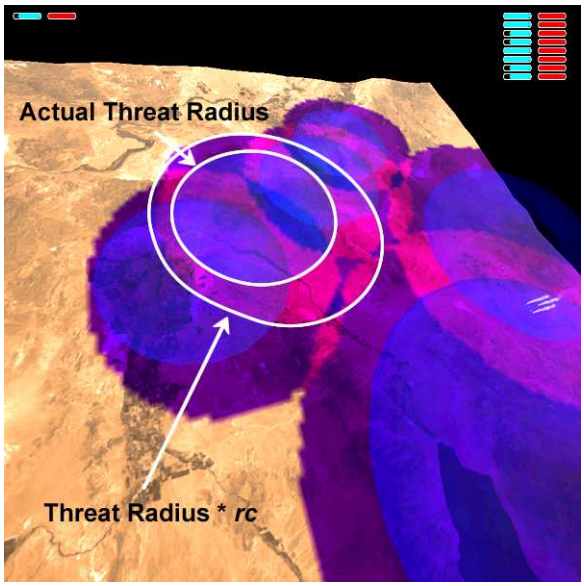

Fig. 5. Routing Map visualization with $rc = 1.4$

B. General Routing Knowledge

A* is shown to always find the optimal route based on its cost functions, in our case the shortest route avoiding known threats. Since our game includes traps, the shortest route is not always desirable. In order to do more interesting routing, we must be able to bias A* towards producing routes that are longer, or more dangerous then those immediately apparent. We do this by modifying the graph A* searches, producing a variety of effects on the kinds of routes produced with relative ease. For example penalizing each voxel based on how far south it is provides a bias that tends to produce north traveling routes, thus producing an overall strategy of attacking from the north. Routing two groups of platforms, one with a southern

bias and one with a northern bias is likely to produce pincer attacks. However the human in our game is trying to avoid confined areas, and to do this we need to modify the voxels in order to identify areas that are confined. If we can increase the cost of those voxels, then the router will avoid those areas. In our representation we identify these confined areas by extending the effective radii of threats when we build the voxel graph. The extension is calculated by a simple multiplication of each radius by a coefficient rc, which determines the kind of routes produced. Figure 5 shows the effect rc has on routing. The inner circle outlines the range of one of the threats, the outer sphere outlines the radius being used by the routing system for this particular mission. At this rc the threats have expanded together and filled the corridor, leading the router to produce green routes.[1] In our mission, rc values less then 1.4 lead to yellow routes while larger rc values lead to green routes. As encoded, rc uses 8 bits to produce a range from 0 to 3. Future work will include additional parameters to deal with the different factors involved in human routing.

C. Encoding

Figure 6 shows how we represent the allocation data as an enumeration of assets to targets. The left box illustrates the allocation of asset A1 to target T3, asset A2 to target T1 and so on. Tabulating the asset to target allocation gives the table in the lower left. By defining the assets to always be in the same sequence we can remove them from the table and then reduce the target id to binary.

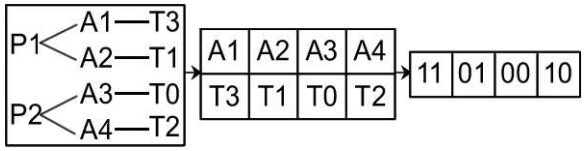

Fig. 6. Allocation Encoding

D. Fitness

Fitness of a plan is calculated by measuring how well the plan achieved each of its goals and summing these measures. Blue's goals are to maximize damage to Red, while avoiding damage to itself. Total damage done is calculated by the equation below.

$$damage(Player) = \sum_{E \subset F} E_v * (1 - E_s)$$

where E is an entity in the game and F is the set of all forces belonging to that side. E_v is the value of that entity. E_s is that entity's probability of surviving the mission. Shorter simple routes are also desirable, so we include a penalty in the fitness based on total distance traveled by platforms. This gives the fitness calculated as shown in Equation 3.

$$fit(plan) = damage(Red) - damage(Blue) - d * c1 \quad (3)$$

[1]This figure looks better in color - you can find these and other color figures at http://www.cs.unr.edu/∼miles/

Where d is the total distance traveled by Blue's platforms and $c1$ is chosen such that $d*c1$ has a 10-20% effect on the fitness. Note that the bias towards shorter routes produces a preference for plans routes that lead into the confined area (trap) for our mission.

IV. Knowledge Acquisition and Application

Imagine playing a game and seeing your opponents do something you had not considered that worked out to great effect. Seeing something new, you are likely to try to learn some of the dynamics of that move so you can incorporate it into your own play and become a more versatile player. Ideally you would like perfect understanding of when and where this move is effective and ineffective, and how to best execute the move under those circumstances. Whether the move is using a combination of chess pieces in a particular way, bluffing in poker, or doing a reaver drop in Starcraft the general idea remains. In order to imitate this process we use a two step approach with case injection. First we learn knowledge from human players by saving their decision making during game play and encoding it in for storage in the case-base. Second we apply this knowledge by periodically injecting these stored cases into GAP's evolving population.

A. Knowledge Acquisition

In this paper, knowledge acquisition takes the form of building a case-base of chromosomes representing past strategies used by human experts. Each strategy should be represented in a general way, so that it can be applied robustly across a variety of missions. rc allows us to represent the knowledge of avoiding confined areas as defined by the expert in our mission.

Actually converting human strategies into chromosomes to save in the case-base is non-trivial. The focus of this research is on knowledge application so the chromosome has been directly engineered by the human player. Representation is key, and we are investigating different encodigs and different methods of translating human decisions to encoded plans.

B. Knowledge Application

Case injection has been shown to increase the search speed of a GA when used across similar problems [1]. It also tends to produce new answers similar to old ones, biasing the search to look in areas that were previously successful. Exploiting this effect gives our GA its learning behavior. By directing our GA to search through strategies similar to those that were effective for humans, we bias it towards producing more human like answers. However we would still like to maintain the flexibility of the search. Consider learning from a human who played a green route, but had a non-optimal allocation. Ideally the GA should keep the green route, but optimize the allocation. Unless the allocation itself was based on some external knowledge (a particular target might seem like a trap), in which case the GA should maintain that knowledge. Perfectly identifying which knowledge to maintain and which to replace is a difficult task even for human players. Our goal

is to bias GAP to probabilistically keep the information that seems most relevant and effective. In this phase of the research we concern ourselves with using GAP to reproduce a useful and easily identifiable aspect of human strategy. During the knowledge acquisition phase, we produce a number of chromosomes representing human plans and store them in a case-base. These plans contain the knowledge to avoid confined areas through a high rc. Subsequently, when playing the game we periodically inject a number of individuals from the case-base in order to bias the search with information from those individuals. The individuals contain information that avoids the confined areas, and through injection we hope to produce green routes in future games. Injection occurs by replacing the worst members of the population with individuals chosen from the case database. A "Probabilistic closest to the best" strategy determines which individuals get injected [2]. Parameters for case-injection include the number of individuals injected, the frequency of injection, as well as specifying when to stop injecting.

V. Results

We present results showing GAP's three major abilities.

1) GAP can form strategies and play the game effectively.
2) Case injection leads to more frequent human-like strategies.
3) Our encoding for potential traps allows this knowledge to be used on different missions.

We also analyze the effect of altering the population size and number of generations on the strength of the biasing provided by case injection.

Unless otherwise stated, GAP uses a population size of 25, two-point crossover with a probability of 0.95, and point mutation with a probability of 0.01. We use elitist selection, where offspring and parents compete for population slots in the next generation [22]. All results are averages over 50 runs.

We first show that GAP can form efficient strategies. GAP plays the mission 50 times, and we graph the average fitness of individuals inside the population against their generation in Figure 7. The graph shows a strong approach toward the optimum. GAP approaches within 5% of optimal allocation and routing 95% of the time. This indicates that GAP can form effective strategies for playing the game to the extent possible from the evaluator.

To test GAP's ability to capture external knowledge and emulate humans we produce green route plans, convert them to chromosomes, and then try to bias our search towards those chromosomes. The human plan is show in white in Figure 8. Converting this to the closest plan representable in our encoding gives the plan shown in green in Figure 8. The plans are not identical because the chromosome does not contain exact routing information. Note the overall fitness difference between these two plans is less then 2%.

The category of routes produced is determined by the values of rc. GAP's ability to produce the human like route (green) is based on the values of rc it chooses. Figures 9 and 10 show the distribution of rc produced by the non-injected genetic

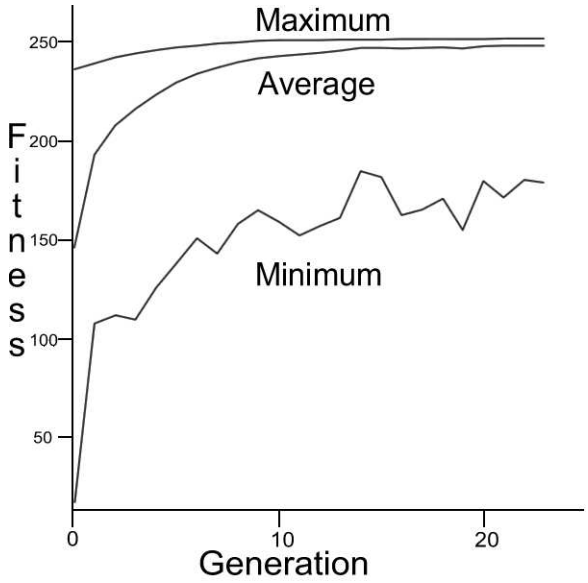

Fig. 7. Best/Worst/Average Individual Fitness as a function of Generation - Averaged over 50 runs.

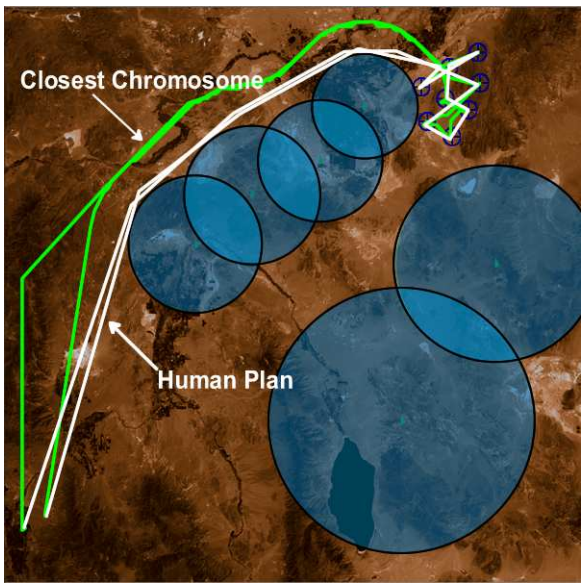

Fig. 8. Plans produced by the Human and Gap

.

algorithm and the case-injected genetic algorithm. Comparing Figure 9 with Figure 10 shows a significant shift in the rc's produced, which leads to a large increase in the number of green routes generated by the case injected GA. Without case injection GAP produced no green routes, using case injection biased GAP to produce 64% green routes, this difference is statistically significant. These results were based on 50 different runs of the system with different random seeds and show that case injection does bias the search towards the human strategy.

Moving to the mission shown in Figure 11 and repeating the process produces the histograms shown in Figures 12

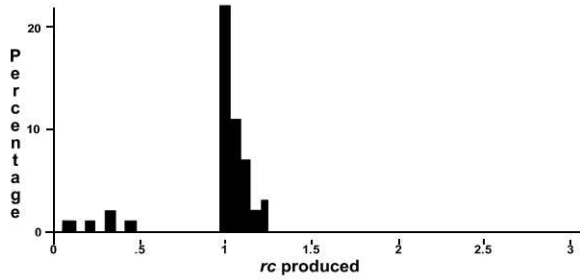

Fig. 9. Histogram of Routing Parameters produced without Case Injection.

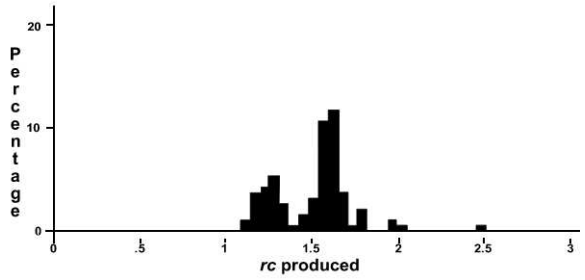

Fig. 10. Histogram of Routing Parameters produced with Case Injection.

and 13. The same effect on rc can be observed even though the missions are significantly different, and even though we use the cases from the previous mission. Our general routing representation allows GAP to learn to avoid confined areas from play by the human expert.

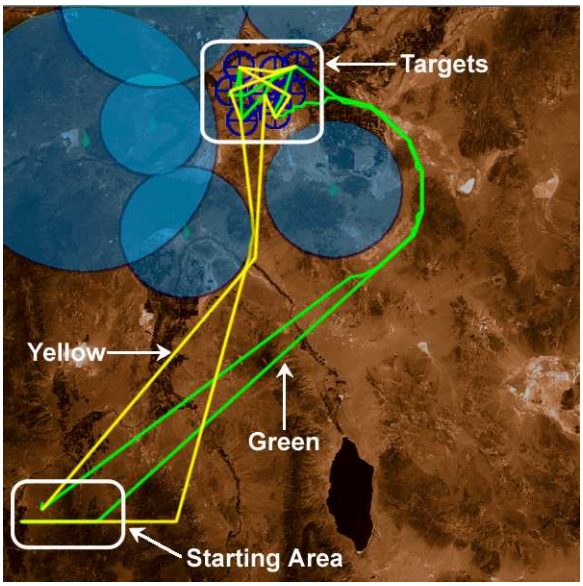

Fig. 11. Alternate Mission

Case injection applies a bias to the GA search, the number and frequency of individuals injected determines the strength of this bias. However the fitness function also contains a term that biases against producing longer routes. As the number of evaluations allotted to the GA is increased, the bias against

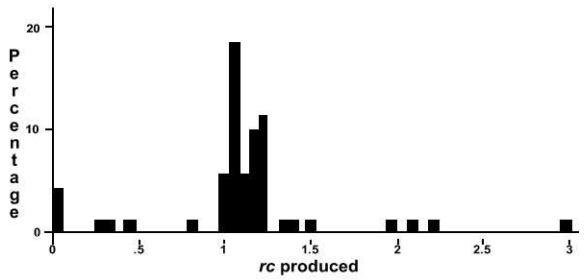

Fig. 12. Histogram of Routing Parameters produced without Case Injection on the Alternate Mission.

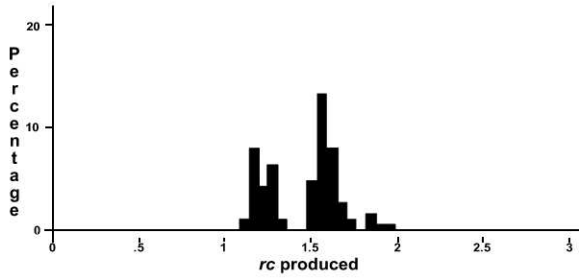

Fig. 13. Histogram of Routing Parameters produced with Case Injection on the Alternate Mission.

longer routes outweighs the bias towards human strategies and fewer green routes are produced. The effect is shown in Figure V.

VI. Conclusions and Future Work

Our original goals were to develop GAP so that it could play the game, and to use case injection to bias its play towards human-like strategies. GAP shows its ability to play and learn, and our results indicate a good first step towards knowledge acquisition and application. We ran many experiments to try to control the strength of the bias provided by case injection. Replacing the entire population with injected individuals led to 86% green routes, but otherwise we were unable to produce more then 40-60% green routes while still optimizing the allocation. We are currently exploring methods for increasing the bias towards using injected materials - in particular artificially changing the fitness of injected individuals and their descendants, so as to better preserve injected material. Expanding the game is also an avenue of major interest. Increasing the complexity of the game will lead to deeper strategic play, providing many areas in which to expand GAP's abilities. Allowing both sides to attack and defend, in conjunction with resource management is the next major phase of the game. We are also interested in representation issues, how to best build the encoding so that it can represent a large variety of strategic ideas likely to be used by humans formulating strategy. Even with good representations reverse-engineering human plans into chromosomes containing relevant and general information presents a formidable problem.

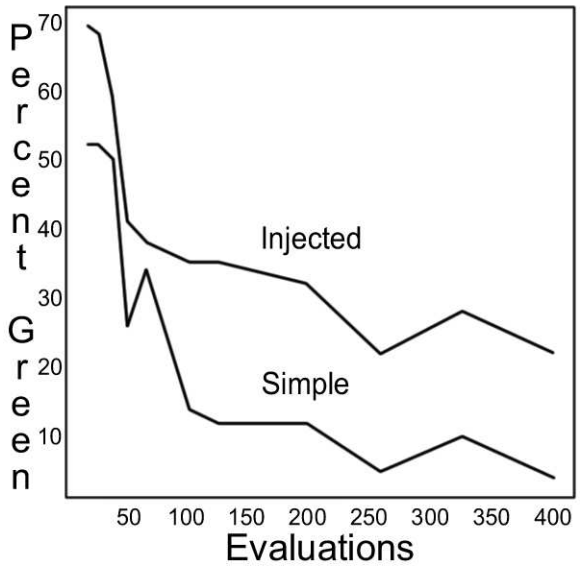

Fig. 14. Number of Evaluations effect on the Percentage Green routes Produced

Acknowledgments

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

REFERENCES

[1] Louis, S.J.: Evolutionary learning from experience. Journal of Engineering Optimization (To Appear in 2004)
[2] Louis, S.J., McDonnell, J.: Learning with case injected genetic algorithms. IEEE Transactions on Evolutionary Computation (To Appear in 2004)
[3] Griggs, B.J., Parnell, G.S., Lemkuhl, L.J.: An air mission planning algorithm using decision analysis and mixed integer programming. Operations Research **45** (Sep-Oct 1997) 662–676
[4] Li, V.C.W., Curry, G.L., Boyd, E.A.: Strike force allocation with defender suppression. Technical report, Industrial Engineering Department, Texas A&M University (1997)
[5] Yost, K.A.: A survey and description of usaf conventional munitions allocation models. Technical report, Office of Aerospace Studies, Kirtland AFB (Feb 1995)
[6] Louis, S.J., McDonnell, J., Gizzi, N.: Dynamic strike force asset allocation using genetic algorithms and case-based reasoning. In: Proceedings of the Sixth Conference on Systemics, Cybernetics, and Informatics. Orlando. (2002) 855–861
[7] Fogel, D.B.: Blondie24: Playing at the Edge of AI. Morgan Kauffman (2001)
[8] 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
[9] 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
[10] Kendall, G., Willdig, M.: An investigation of an adaptive poker player. In: Australian Joint Conference on Artificial Intelligence. (2001) 189–200
[11] Samuel, A.L.: Some studies in machine learning using the game of checkers. IBM Journal of Research and Development **3** (1959) 210–229
[12] Blizzard: Starcraft (1998, www.blizzard.com/starcraft)
[13] Cavedog: Total annihilation (1997, www.cavedog.com/totala)
[14] Inc., R.E.: Homeworld (1999, homeworld.sierra.com/hw)
[15] Laird, J.E.: Research in human-level ai using computer games. Communications of the ACM **45** (2002) 32–35
[16] Laird, J.E., van Lent, M.: The role of ai in computer game genres (2000)
[17] Laird, J.E., van Lent, M.: Human-level ai's killer application: Interactive computer games (2000)
[18] Tidhar, G., Heinze, C., Selvestrel, M.C.: Flying together: Modelling air mission teams. Applied Intelligence **8** (1998) 195–218
[19] Serena, G.M.: The challenge of whole air mission modeling (1995)
[20] McIlroy, D., Heinze, C.: Air combat tactics implementation in the smart whole air mission model. In: Proceedings of the First Internation SimTecT Conference, Melbourne, Australia, 1996. (1996)
[21] Stout, B.: The basics of a* for path planning. In: Game Programming Gems, Charles River media (2000) 254–262
[22] Eshelman, L.J.: The CHC adaptive search algorithm: How to have safe search when engaging in nontraditional genetic recombination. In Rawlins, G.J.E., ed.: Foundations of Genetic Algorithms-1. Morgan Kauffman (1991) 265–283