Learning to Play Like a Human: Case Injected Genetic Algorithms for Strategic Computer Gaming

Sushil J. Louis[a] and Chris Miles[a]

[a] Evolutionary Computing Systems LAB, University of Nevada, Reno - 89557, {miles,sushil}@cse.unr.edu

ABSTRACT

We use case injected genetic algorithms to learn how to competently play computer strategy games that involve long range planning across complex dynamics. Imperfect knowledge presented to players requires them adapt their strategies in order to anticipate opponent moves. We focus on 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 force planning game. By incorporating case injection into a genetic algorithm, we show methods for incorporating general knowledge elicited from human players into future plans. In effect allowing the GA to take important strategic elements from human play and merging those elements into its own strategic thinking. 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.

Keywords: genetic algorithms, case-injection, decision support, training

1. INTRODUCTION

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

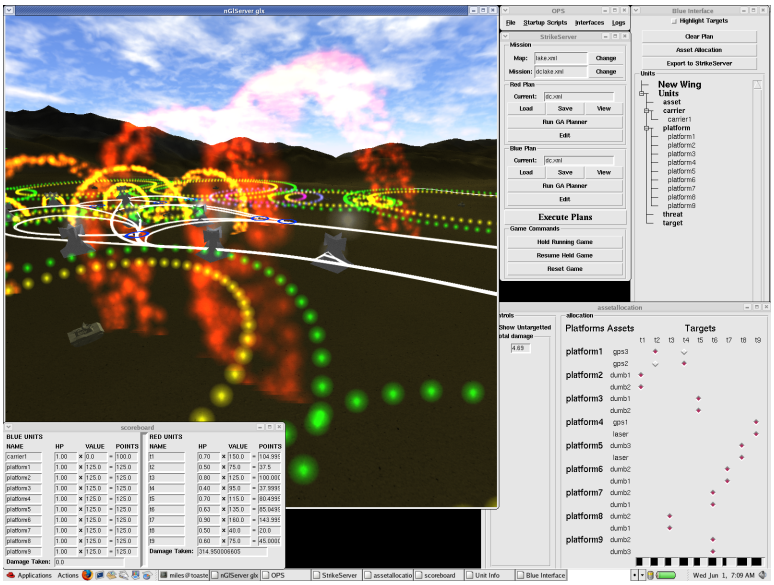

Figure 1. Game Screen-shot.

allocation strategies, but these strategies may may 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 (gained through their experience) when producing strategies for the real-world problem underlying the game. Acquiring and incorporating knowledge from the way these humans play should allow us to carry over some of this external knowledge into our own play. Results show that case injection combined with a flexible

representation biases the genetic algorithm toward producing strategies similar to those learned from human players. Beyond playing similarly on the same mission, the genetic algorithm can use strategic knowledge across a range of similar missions to continue to play as the human would. Figures 1 and 2 show screenshots of the "game" we are designing and its interface.

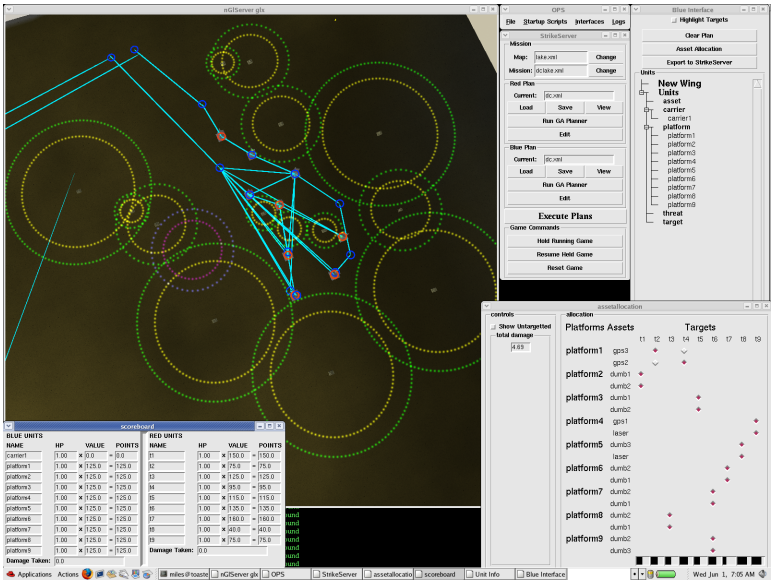

Figure 2. Game Interface Screen-shot.

The next section introduces the "game" we are working with and describes relevant prior work in evolutionary computing and optimization. We introduce the Case-Injected Genetic Algorithm (CIGAR) and show how CIGAR can extract and use knowledge from a subject matter expert. Section 3 describes our test problem - the mission that is being run and Section 4 provides results and a short discussion. The last section provides conclusions and future research.

2. THE GAME

Strike force asset allocation 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, therefore enticing more people to play the "game" and so that we can acquire player knowledge from them. The game involves two sides: Blue and Red, Blue allocates a set of platforms (aircraft) to attack Red's targets (buildings) while Red has defensive installations (threats). These threats complicate 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 and Red's ability to play popups can affect its planning. 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 platforms and their assets (weapons) as efficiently as possible in order to destroy the targets while minimizing risk. Many factors determine risk. The platform's route, the effect of accompanying wingmen, weather, and the presence of threats around chosen targets. GAP uses genetic algorithms to develop strategies for the attacking strike force, including flight plans and weapon targeting for all available aircraft. An attacking strategy is an allocation of assets (weapons) to aimpoints on targets - as such GAP uses the genetic algorithm to quickly search through the space possible allocations for an optimal asset allocation. Genetic algorithms are good at asset allocation problems and are thus well suited to playing the game with game decisions modeled as asset allocation optimization.

When confronted with popups, GAP responds by replanning with the case-injected genetic algorithm to produce a new plan of action. Case-injected genetic algorithms use a case-base of past problem solving experience as a long-term memory store in a genetic algorithm based machine learning systems.

2.1. Case-injected Genetic Algorithms

Typically, a genetic algorithm randomly initializes its starting population so that the GA can proceed from an unbiased sample of the search space. However, since problems do not usually exist in isolation, we expect a system deployed in an industrial setting to confront a large number of problems over the system's lifetime. Many of these problems may be similar. In this case it makes little sense to start a problem solving search attempt from scratch when previous searches may have yielded useful information about the problem domain. Instead, periodically injecting a genetic algorithm's population with *relevant* solutions (we describe what we mean by relevant later) or partial solutions from previously solved similar problems can provide information (a search bias) that reduces the time taken to find an acceptable solution. Our approach borrows ideas from case-based reasoning (CBR) in which old problem and solution information, stored as cases in a case-base, helps solve a new problem.[10,17,20] In our system, the data-base, or case-base, of problems and their solutions supplies the genetic problem solver with a long term memory. The system does not require a case-base to start with and can bootstrap itself by learning new cases from the genetic algorithm's attempts at solving a problem.

The case-base does what it is best at – memory organization; the genetic algorithm handles what it is best at – adaptation. The resulting combination takes advantage of both paradigms; the genetic algorithm component delivers robustness and adaptive learning while the case-based component speeds up the learning process.

The CIGAR system presented in this paper can be applied in a number of domains from computational science and engineering to operations research. We concentrate on asset allocation for our game in this paper.

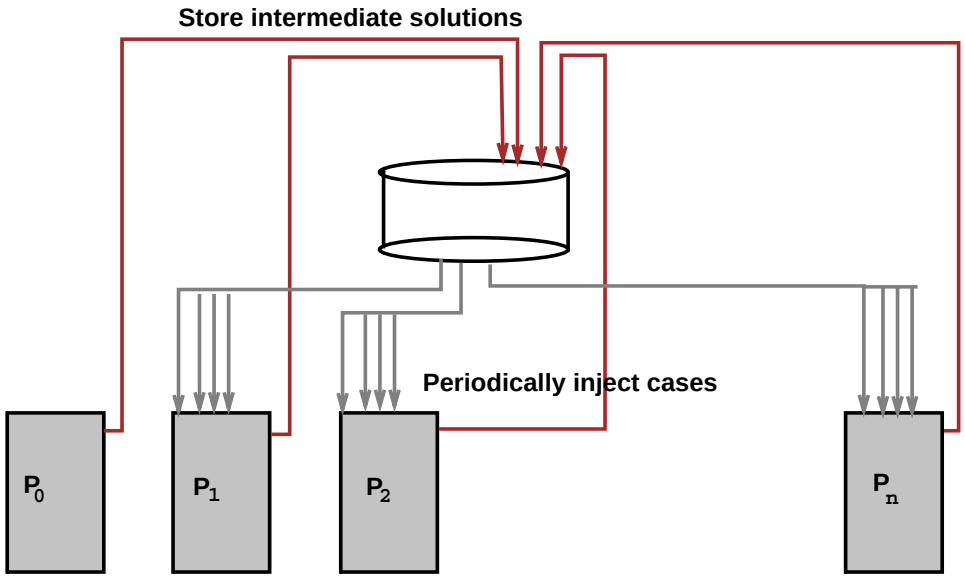

Figure 3. The CIGAR game playing system

Figure 3 depicts the genetic algorithm and its interaction with the case-base in solving a sequence of problems (game decisions). As the genetic algorithm searches through the space of possible allocations on problem P_0, it periodically saves high fitness individuals to the case-base. Cases in CIGAR are individuals from a genetic algorithm's population along with some ancillary information like a unique case identifier, the individual's fitness, pointers to the individual's parents, and a problem identifier. These individuals, or cases, encode candidate solutions or partial solutions to the game's underlying resource allocation problem. The case-base also obtains cases from human game-play - we reverse-engineer human player generated allocations (game-play decisions) into the encoding used by the genetic algorithm and store these cases in the case-base. On subsequent missions in the

game $(P_1, P_2, ...P_n)$, the system injects appropriate cases into the genetic algorithm player's evolving population thus biasing search towards solutions similar to injected cases. If injected cases come from human players, the knowledge encapsulated in these cases biases GAP. Alternatively, if cases come from previous GAP attempts at playing the game, then the system learns from its own previous game playing experience. The system does not require a case-base to start with and can bootstrap itself by learning new cases from the genetic algorithm's attempts at solving a problem.

Beyond producing near optimal strategies we would like CIGAR to learn to produce 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 who incorporates strategies taken from a range of human experts. Instead of injecting cases from past problem solving attempts we inject cases that are the result of humans playing the game. Injecting cases that arise from human decision making during gameplay should bias CIGAR into producing solutions that are similar to human solutions. 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. We show that GAP can use information taken from a human player, even when confronted with a different mission, to continue to play strategies closer to the style of that human then GAP's non injected counterpart. That is, GAP can generalize its solutions.

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.[4, 11, 14, 24] A large body of work exists in which evolutionary methods have been applied to games.[3, 6, 16, 18, 19] However the majority of this work has been applied to board, card, and other well defined games which have many differences from popular real time strategy (RTS) computer games such as Starcraft, Total Annihilation, and Homeworld.[1, 2, 5] Entities in our game exist and interact over time in continuous three dimensional space. Sets of algorithms control these entities, seeking to fulfill goals set by the player leading them. This adds a level of abstraction not found in those traditional games. In most of these computer games, players not only have incomplete knowledge of the game state, but even identifying the domains of this incomplete knowledge is difficult. John Laird[7–9] 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,[15, 21, 23] the purpose of our game is not to provide a perfect reflection of reality however, but to both provide a platform for research in strategic planning and to have fun.

3. MISSION AND ENCODING

The mission being played is shown in Figure 4. 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.

To play the game, GAP must therefore produce routing data for each of Blue's platforms. Figure 5 shows how routes are built using the A* algorithm.[22] A* builds routes between locations that platforms wish to visit, generally the starting airbase and targets they are attacking. The A* router finds the cheapest route, where cost here is a function of length and risk and leads to a preference for the shortest routes that avoid threats.

We parameterize A* in order to represent and produce routes that are not dependent on geographical location and that have specific characteristics. For example, to avoid traps, GAP must be able to specify that it wants to avoid areas of potential danger. In our game, traps are most effective in areas confined by other threats. If we artificially inflate threat radii, threats expand to fill in potential trap corridors and A* produces routes that go around these expanded threats. We thus introduce a parameter, rc that encodes threats' effective radii.

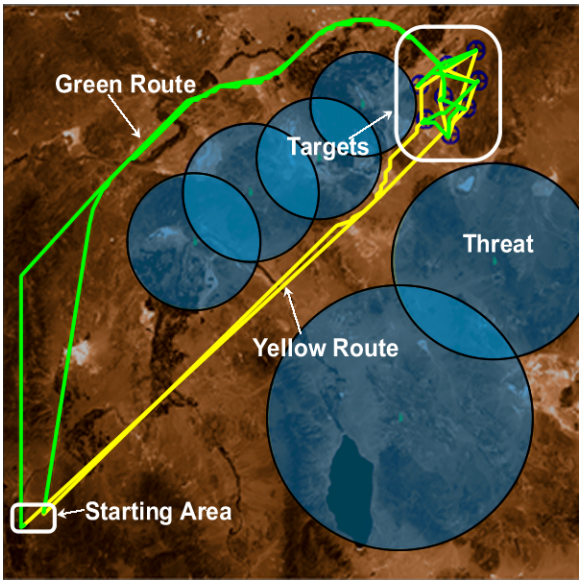

Figure 4. Mission scenario

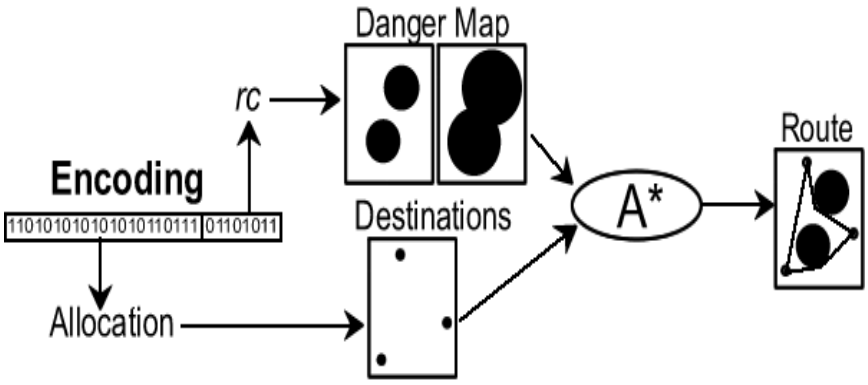

Figure 5. How Routes are Built From an Encoding.

Larger rc's expand threats and fill in confined areas, smaller rc's lead to more direct routes. Figures 6 and 7 shows rc's effect on routing, as rc increases, A* produces routes that avoid the confined area. In our scenarios,

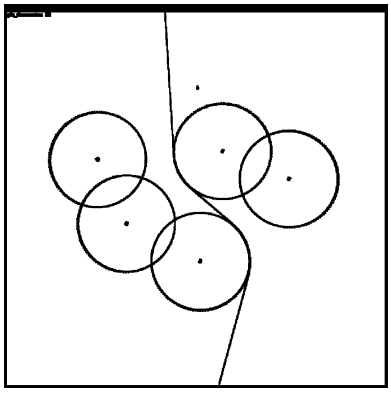

Figure 6. Routing With $rc = 1.0$

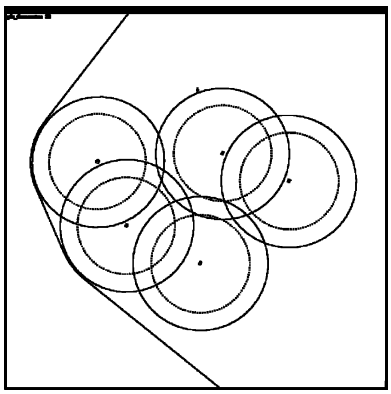

Figure 7. Routing With $rc = 1.3$

values of $rc < 1.0$ produce grey routes, values with $1.0 < rc < 1.35$ produce direct black routes, and values of $rc > 1.35$ produce white trap-avoiding routes. rc is limited currently to the range $[0, 3]$ and encoded with eight (8) bits at the end of our chromosome. We encoded a single rc for each plan but are investigating the encoding of rc's for each section of a route. Note that this representation of routes is location independent. We can store and re-use values of rc that have worked in different terrains and different locations to produce more direct or indirect routes.

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, as it is the shortest. The human expert however, understands the potential for danger in the corridor because of likelyhood of 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.

4. RESULTS

Parameterizing our router allows us to produce routes of different types, incorporating this parameter into the chromosome then allows individuals to contain information about not just what weapons to use and which targets to attack, but general information about how to reach those targets. Specifically, referring to Figure 4 we want

to learn the parameter for green routes and overcome the algorithm's preference for short routes in order to avoid traps. By injecting cases (from humans) that incorporate a preference for green routes we can change the proportion of green routes produced by our system, despite such solutions having lower than evaluator-optimal fitness. Furthermore by artificially inflating the fitness of injected solutions and their descendants in the system's population, we can control the proportion of routes produced that avoid potential traps.

Figure 8 shows these proportions in terms of our routing parameter, *rc*.

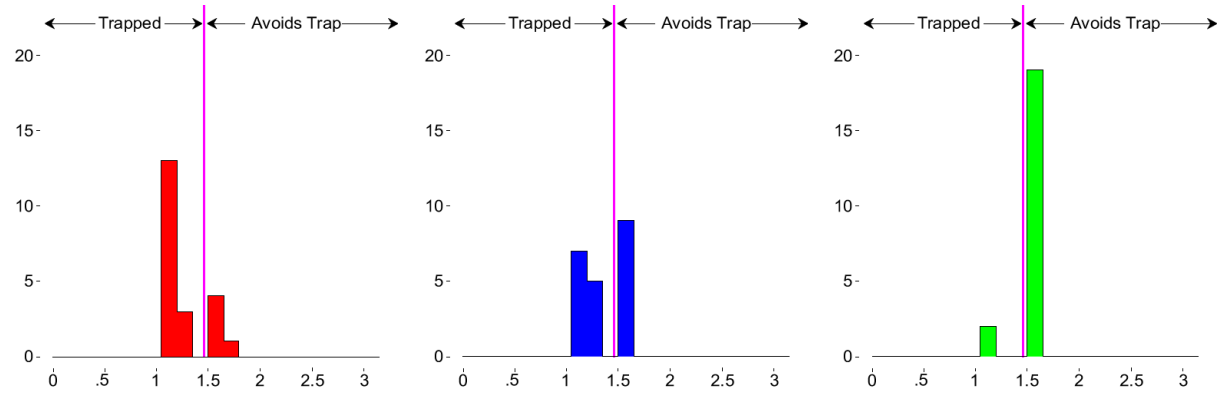

Figure 8. Distribution of RC values. Left: GA alone. Middle: With case injection. Right: Case Injection and fitness biasing.

The figure shows that GAP using CIGAR is able to learn to avoid traps and prefer green routes over yellow routes. The subject matter expert (human player) does not need to do anything beyond "playing" the game for this to occur. That is, the human player simply solves the problem using our game's interface - the solution is then converted to a case and stored in the case-base. Thus knowledge extracted from a subject matter expert can be extracted and stored in the case-base for future re-use. When injected into GAP's CIGAR, GAP produces solutions that show a preference for trap-avoiding routes. That is, CIGAR extracts and uses knowledge from subject matter expert problem solving to help solve new problems. Although this paper only lists results on one mission we have a number of other supporting results at http://www.cse.unr.edu/~sushil/.

5. CONCLUSIONS AND FUTURE WORK

We started out wishing to investigate knowledge acquisition and incorporation (from humans) to bias genetic algorithm search to model human decision making. Knowledge acquisition is generally considered a difficult Artificial Intelligence (AI) problem and we tackle this using case-injected genetic algorithms. 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 previous problem solving experience.[12,13] In this work, 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 by acquiring and 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. The genetic algorithm searches for an optimal strategy while case injection biases it to contain elements from strategies used by humans. Used in conjunction we hoped to produce near optimal strategies that incorporate important information external to the evaluation function.

The results show that our approach can learn from human subject matter experts and produce near-optimal solutions to the asset allocation problem. Specifically, we show that a case-injected genetic algorithm learns trap avoidance from expert game-play and that the system can generalize this trap avoiding knowledge in a location (mission) independent way.

In the future, we would also like to incorporate dynamics into the game without the expense of simulating them. Consider a game involving armies doing battle, both players periodically 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 these game dynamics from watching humans play, it should deepen the feeling of strategy involved in playing the game.

Our Genetic Algorithm Player (GAP) should be able to function both as a decision aid and as a trainer. 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 less adept human players without the expense of individual training with experts. We also want to use GAP for decision support, whereby GAP provides suggestions and alternative strategies to humans actively playing the game. Producing strategies that are both near optimal strategies and compatible with those being considered by human players, should help maximize the positive effect on the decision making process.

These roles require GAP to play with objectives in mind besides that of winning – objectives that would be difficult to quantify inside the evaluator. As humans function effectively in these regards, learning from them should help GAP better fulfill these responsibilities. Preliminary results indicate that case injection seems to be a promising approach towards acquiring knowledge and biasing genetic search toward human preferred solutions.

Acknowledgment

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

REFERENCES

1. Blizzard. Starcraft, 1998, www.blizzard.com/starcraft.
2. Cavedog. Total annihilation, 1997, www.cavedog.com/totala.
3. David B. Fogel. *Blondie24: Playing at the Edge of AI*. Morgan Kauffman, 2001.
4. B. J. Griggs, G. S. Parnell, and L. J. Lemkuhl. An air mission planning algorithm using decision analysis and mixed integer programming. *Operations Research*, 45(5):662–676, Sep-Oct 1997.
5. Relic Entertainment Inc. Homeworld, 1999, homeworld.sierra.com/hw.
6. Graham Kendall and Mark Willdig. An investigation of an adaptive poker player. In *Australian Joint Conference on Artificial Intelligence*, pages 189–200, 2001.
7. John E. Laird. Research in human-level ai using computer games. *Communications of the ACM*, 45(1):32–35, 2002.
8. John E. Laird and Michael van Lent. Human-level ai's killer application: Interactive computer games, 2000.
9. John E. Laird and Michael van Lent. The role of ai in computer game genres, 2000.
10. David B. Leake. *Case-Based Reasoning: Experiences, Lessons, and Future Directions*. AAAI/MIT Press, Menlo Park, CA, 1996.
11. V. C-W. Li, G. L. Curry, and E. A. Boyd. Strike force allocation with defender suppression. Technical report, Industrial Engineering Department, Texas A&M University, 1997.
12. Sushil J. Louis. Evolutionary learning from experience. *Journal of Engineering Optimization*, To Appear in 2004.
13. Sushil J. Louis and John McDonnell. Learning with case injected genetic algorithms. *IEEE Transactions on Evolutionary Computation*, To Appear in 2004.
14. Sushil J. Louis, John McDonnell, and N. Gizzi. Dynamic strike force asset allocation using genetic algorithms and case-based reasoning. In *Proceedings of the Sixth Conference on Systemics, Cybernetics, and Informatics. Orlando*, pages 855–861, 2002.
15. D. McIlroy and C. Heinze. Air combat tactics implementation in the smart whole air mission model. In *Proceedings of the First Internation SimTecT Conference, Melbourne, Australia, 1996.*, 1996.

16. Jordan B. Pollack, Alan D. Blair, and Mark Land. Coevolution of a backgammon player. In Christopher G. Langton and Katsunori Shimohara, editors, *Artificial Life V: Proc. of the Fifth Int. Workshop on the Synthesis and Simulation of Living Systems*, pages 92–98, Cambridge, MA, 1997. The MIT Press.
17. C. K. Riesbeck and R. C. Schank. *Inside Case-Based Reasoning*. Lawrence Erlbaum Associates, Cambridge, MA, 1989.
18. Christopher D. Rosin and Richard K. Belew. Methods for competitive co-evolution: Finding opponents worth beating. In Larry Eshelman, editor, *Proceedings of the Sixth International Conference on Genetic Algorithms*, pages 373–380, San Francisco, CA, 1995. Morgan Kaufmann.
19. A. L. Samuel. Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development*, 3:210–229, 1959.
20. R. C. Schank. *Dynamic Memory: A Theory of Reminding and Learning in Computers and People*. Cambridge University Press, Cambridge, MA, 1982.
21. Graeme Murray Serena. The challenge of whole air mission modeling, 1995.
22. Bryan Stout. The basics of a* for path planning. In *Game Programming Gems*, pages 254–262. Charles River media, 2000.
23. Gil Tidhar, Clinton Heinze, and Mario C. Selvestrel. Flying together: Modelling air mission teams. *Applied Intelligence*, 8(3):195–218, 1998.
24. K. A. Yost. A survey and description of usaf conventional munitions allocation models. Technical report, Office of Aerospace Studies, Kirtland AFB, Feb 1995.