Trap Avoidance in Strategic Computer Game Playing with Case Injected Genetic Algorithms

Chris Miles, Sushil J. Louis, and Rich Drewes

Evolutionary Computing Systems Lab
Department of Computer Science
University of Nevada
Reno - 89557
{miles,sushil,drewes}@cs.unr.edu

Abstract. We use case injected genetic algorithms to learn to competently play computer strategy games. Such games are characterized by player decisions made in anticipation of opponent moves with imperfect knowledge of the game state. Within the broad goal of developing effective and general methods for anticipatory play, this paper investigates anticipation in the context of trap avoidance in an immersive, 3D strike planning game. Case injection is used for learning through experience, providing methods for learning likely opponent moves and acting in anticipation of them. Results show that with an appropriate representation case injection is effective at extracting knowledge from past games, knowledge containing likely opponent traps, and biasing the genetic algorithm towards producing plans containing that knowledge, avoiding the trap while still carrying out the mission effectively.

1 Introduction

The computer gaming industry is now bigger then the movie industry and both gaming and entertainment drive research in many aspects of computer hardware and software. However AI research has been mainly interested in games such as checkers and chess, games with many differences from popular computer games like Starcraft and Counter-Strike. These games take place in a virtual world involving long-term and reactive planning, they involve countless varieties of player interactions, and they provide an immersive, fun experience. We can pose many business, training, planning, and scientific problems as games in which player decisions determine the final solution. Systems that play well in such a game, correspond closely with systems that are effective in the "real" world.

This paper applies a case-injected genetic algorithm that combines genetic algorithms with case-based reasoning to a domain context modeled by computer games [1]. The genetic algorithm "plays" the game by solving the underlying decision making problems. Specifically, we develop and use a strike force asset allocation game, which maps to a broad category of resource allocation problems in industry, as our test problem. Strike force planning consists of allocating a collection of strike assets on flying platforms to a set of ground based targets

and threats. The problem is dynamic; weather and other environmental factors affect asset performance, unknown threats can popup, and new targets can be assigned. These complications as well as the varying effectiveness of assets on targets produce a non-linear space suitable to genetic and evolutionary computing approaches.

The idea behind a case-injected genetic algorithm is that as the genetic algorithm iterates over a problem it selects members of its population and stores them, forming a case base of past individuals with useful knowledge to use on future problems. Periodically when similar scenarios, the system injects appropriate cases from the case base, merging strategies previously successful on related situations into the evolving population.

Case Injection is first used when the dynamic nature of the game requires players to modify their strategies in order to respond to situational changes. We have shown that case-injected genetic algorithms learn to increase performance with experience at solving similar problems [1–5]. This implies that a case-injected genetic algorithm should quickly produce new plans in response to changing game dynamics. Beyond purely responding to immediate scenario changes we use case injection to produce plans that anticipate opponent moves, resulting in better positions from which to respond to similar changes in situation.

In the rest of the paper we define the game, the particular scenario being played, and the trap being encountered. We outline GAP's architecture; detailing the use of genetic algorithms, the encoding of strategy, the routing system, and the incorporation of case injection. Section 6 presents results showing that GAP can effectively play the game in the absence of the trap, and that GAP can quickly re-plan in the face of changing game dynamics. Results also show the effectiveness of using case injection towards learning information from past games, allowing GAP to learn from experience to avoid potential traps. The last section presents our conclusions and current directions for future work.

2 The Strike Planning Game

The strike planning game being played is far from a commercial game such as Starcraft, but it is built on top of a professional game engine and contains many of the complexities of such a game. Our game involves two sides: Blue and Red, both seeking to allocate their respective resources to maximize the damage dealt to their opponent, while minimizing the damage they receive.

Blue plays by allocating aircraft (platforms) and the weapons (assets) they carry to destroy Red's buildings (targets). This involves determining which assets to use on which targets, and how to route the platforms in order to accomplish their objectives while minimizing the risk presented to them.

Red plays by placing defensive installations (threats) to defend its targets, threats that attack any platform coming within range. Potential threats and targets can also "pop-up" on Red's command in the middle of a mission, allowing a range of strategic options. By cleverly locating threats Red can feign

vulnerability and lure Blue into a deviously located popup trap, or keep Blue from exploiting such a weakness out of fear of a trap. The scenario in this paper involves Red presenting Blue with a corridor of easy access to unprotected targets, a corridor containing a popup threat.

In this paper, Red's play is human scripted while a Genetic Algorithm Player (GAP) plays Blue. GAP develops strategies for the attacking strike force, including flight plans and weapon targeting for all available aircraft. When confronted with popups or other changes, GAP responds by replanning with the genetic algorithm, producing a new plan in response to the new situation. Beyond purely responding to immediate changes we use case injection to produce plans that anticipate future opponent moves, providing GAP with better positions from which to respond to changes when they occur. In this game and scenario, our primary goal is not just to improve the GA's performance through case-injection such as in previous work, but to bias the search using knowledge previously acquired. This knowledge is external to the game simulation and the evaluation of fitness, in particular we desire GAP to gain knowledge allowing it to avoid potential traps.

2.1 Previous Work

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 [6] 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 [7] converts a nonlinear programming formulation into a MIP problem. Yost [8] provides a survey of the work that has been conducted to address the optimization of strike allocation assets. Louis [9] applied case injected genetic algorithms to strike force asset allocation. A large body of work exists in which evolutionary methods have been applied to games [10–14]. 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[15–17]. Chess, checkers and many others use entities (pieces) that have a limited space of positions (such as on a board) and restricted sets of actions (orthogonal movements, simplified combat). Players in these games have well defined roles and the domain of knowledge available to each player is well 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 trying to meet goals outlined by players, this adds a level of abstraction not found in more traditional games. In most such computer games, players have incomplete knowledge of the game state and even the domain of this incomplete knowledge is often difficult to determine. John Laird [18–20] 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 [21–23], however these attempt to model reality while ours is designed to provide a platform for research in strategic planning and knowledge acquisition, and to have fun. The next section describes the scenario being used in our experiments.

3 The Scenario

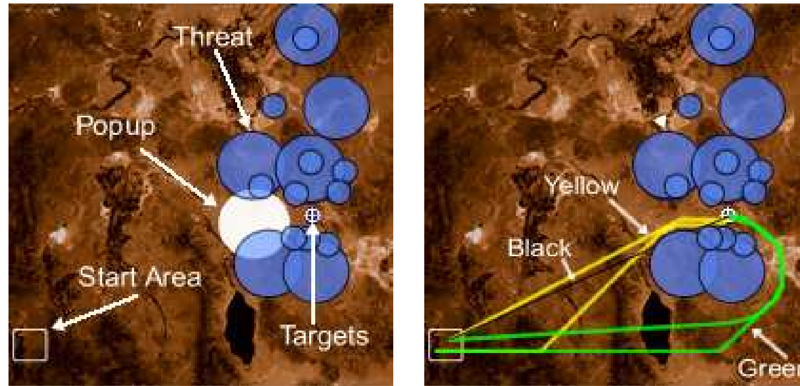

Fig. 1. Left: The Scenario Right: Route Categories

Figure 1-Left shows an overview of our test scenario – chosen to be simple and easily analyzable but to still encapsulate the dynamics of traps and anticipation. We have found that the system scales well into playing more complicated scenarios, so this provides a simple test bed from which to analyze GAP's effectiveness. The scenario takes place in Northern Nevada and California, Lake Tahoe is visible south of the popup. Red has four targets on the right hand side of the map denoted by the white cross-hair. Our point of view is at a significant distance from the targets, so they appear as only a single cross-hair. Red has twenty two (22) threats placed to defend the targets and the translucent blue hemispheres show the effective radii of these threats. Red also has the potential to play the popup threat shown in white near the middle, the location being chosen to trap platforms venturing into the corridor.

Blue has eight platforms, all of which start within in the lower left hand corner. Each platform has one weapon, with three classes of weapons being distributed among the platforms. A weapon-target effectiveness table determines the effectiveness of each weapon against each target. Each of the eight weapons can be allocated to any of the four targets, giving $4^8 = 2^{16} = 64k$ allocations. This space is exhaustively search-able, but more complex scenarios quickly become intractable.

In this scenario GAP produces routes that can be categorized as shown in Figure 1-Right.

1. Black - Flies inside the perimeter of known threats.
2. Yellow - Flies into the corridor between threats in order to reach the targets.
3. Green - Flies around the threats, attacking the targets from behind.

Black routes expose platforms to unnecessary risk, thus strategies containing them receive very low fitness and are quickly removed from the population. The naively optimal strategy contains yellow routes, as they are the shortest most direct routes to the target still avoiding known danger. Strategies containing green routes are preferable because of the possibility of unknown danger, as they will avoid the likely trap waiting within the corridor. In order to search for good routes and allocations, GAP must be able to compute and compare the fitness for potential plans. Computing this fitness is dependent on the representation of entities states inside the game, and our way of representing this state is rather unusual so we next detail it.

3.1 Probabilistic Health Metrics

In many games, entities (platforms, threats and targets) posses hit-points which represents their ability to take damage. Each attack removes a number of hit-points and when reduced to zero hit-points the entity is destroyed and cannot participate further. However, realistically weapons have a hit or miss effect, entirely destroying an entity or leaving it functional. A single attack may be effective while multiple attacks may have no effect. Although more realistic, this introduces a degree of stochastic error into the game. In the worst case, evaluating a individual plan can result in outcomes ranging from total failure to perfect success making it difficult to compare two plans based on a single evaluation. Lacking a good comparison it is difficult to search for an optimal strategy. By taking a statistical analysis of survival we can achieve better results.

Consider the state of each entity at the end of the mission as a random variable. Comparing the expected values for those variables becomes an effective means to judge the effectiveness of a plan. These expected values can then be estimated by executing each plan a number of times 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. Being attacked no longer destroys objects and removes them from the game, it reduces their probability of survival according to Equation 1 below.

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

E is the entity being considered, a platform, target, or threat. $S(E)$ is the probability of survival of entity E after the attack. $S_{t_0}(E)$ is probability of survival of

E up until the attack and $D(E)$ is the probability of that entity being destroyed by the attack and is given by equation 2 below.

$$D(E) = S(A) * E(W)) \tag{2}$$

Here, $S(A)$ is the attackers probability of survival up until the time of the attack and $E(W)$ is the effectiveness of the attackers weapon as given in the weapon-entity effectiveness matrix. This method gives us the true expected values of survival for all entities in the game within one run of the game, producing a representative, non-stochastic, and easily comparable value for the fitness of that plan. These values form a smoother gradient for search, and are consistently reproducible. This technique becomes impractical when entities engage in more complicated relationships, but it is effective at this stage of research.

The gaming system's architecture reflects the flow of action in the game and is described next.

4 System Architecture

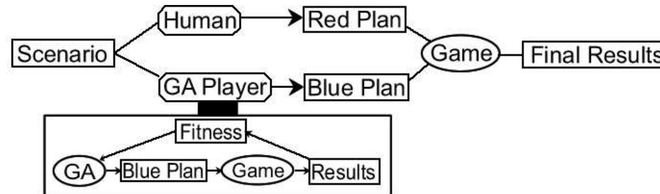

Fig. 2. System Architecture.

Figure 2 outlines our system's architecture. Starting at the left, Red and Blue, human and GAP, are presented with the scenario and given time to prepare their strategy. GAP works by applying the genetic algorithm to the underlying resource allocation problem, using a the routing system to produce final plans. We chose the best plan produced by the GA in the time available to play against Red. These plans then execute, and during execution Red can script the emergence of a popup threat. If a popup is detected by GAP, the genetic algorithm re-plans and switches to a new plan in response to the popup.

4.1 Routing

To play the game GAP must produce routing data for each of Blue's platforms. Figure 3 shows how routes are built using the A* algorithm [24]. This is done by determining the locations the platforms must ultimately reach (target locations), and then using an A* router to find the best paths between each destination.

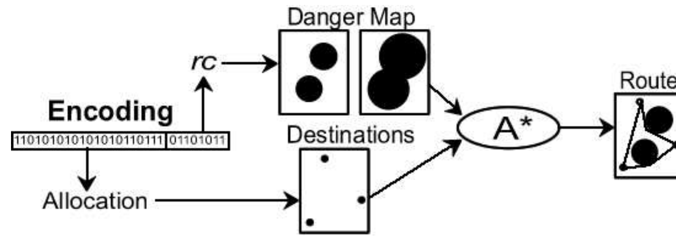

Fig. 3. How Routes are Built From an Encoding.

In order to avoid traps the routing system must be somehow parameterized to avoid areas with particular characteristics. We note that traps are most effective in areas confined by other threats, especially the trap in our scenario. To avoid those areas we introduce a single coefficient into our routing system. When producing routes, threats have their effective radii multiplied by a coefficient rc, with larger rc's threats expand and fill in confined areas. A* then routes around those confined areas. Combined with case injection, rc allows GAP to learn coefficients that avoid traps and re-use them in new scenarios. In our scenario $rc < 1.0$ produce black routes, $1.0 < rc < 1.35$ produce yellow routes and $rc > 1.35$ produce green routes. rc is limited to the range $[0, 3]$ and encoded with 8 bits onto the end of the chromosome. We are encoding a single rc for each plan, future work may include rc's for each section of routing contained in the plan.

4.2 Encoding

The majority of the encoding holds the asset to target allocation, with rc encoded on the end as detailed above. Figure 4 shows how we represent the allocation data as an enumeration of assets to targets. The scenario involves two platforms (P1, P2), each with a pair of assets, attacking four targets. The left box illustrates the allocation of asset A1 on platform P1 to target T3, asset A2 to target T1 and so on. Tabulating the asset to target allocation gives the table in the center. Letting the position denote the asset and reducing the target id to binary then produces a binary string representation for the allocation.

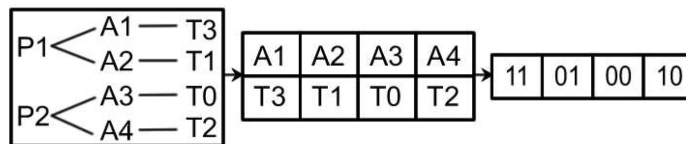

Fig. 4. Allocation Encoding

4.3 Fitness

The Fitness of a plan is calculated by summing how well it achieved its goals. Blue's goals are to maximize damage done to red targets, while minimizing damage done to its platforms. Shorter simpler routes are also desirable, so we include a penalty in the fitness function based on the total distance traveled. This gives the fitness calculated as shown in Equation 3

$$fit(plan) = TotalDamage(Red) - TotalDamage(Blue) - d * c \tag{3}$$

d is the total distance traveled by Blue's platforms and c is chosen such that $d * c$ has a 10-20% effect on the fitness ($fit(plan)$). Total damage done is calculated below.

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

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 E, while E_s is the probability of survival for entity E.

5 Avoiding Traps with Case-Injection

We address the problem learning from experience to avoid traps with a two part approach. First we learn from experience where traps are likely to be, then we apply that knowledge and avoid potential traps in the future. Case-Injection provides an implementation of these steps: building a case-base of individuals from past games stores important knowledge, the injection of those individuals applies the knowledge towards future search.

GAP records games played against opponents, running offline after the game in order to determine the optimal way to win that game. The evaluator now contains knowledge about future opponents moves. Allowing the search to progress towards the optimal strategy, a strategy that acts in anticipation of those opponent moves. GAP saves individuals from this search into the case-base, building a library of knowledge about effective anticipatory play. When faced with other opponents, GAP then injects individuals from the case-base, biasing the current search towards containing this learned anticipatory knowledge.

In this paper GAP first plays the scenario, likely picking a yellow route and falling into Red's trap. Afterwords GAP replays the game, including Red's trap into the evaluator. Yellow routes then receive poor fitness, and GAP searches towards the optimal green route. Saving individuals to the case-base from this search stores a cross-section of plans containing "trap avoiding" knowledge.

The process gives a case-base of individuals containing important knowledge about how we should play, but how can we use that knowledge in order to play smarter in the future? Case Injection has been shown [2] to increase the search speed and the quality of the final solution produced by a GA working on a similar problem. It also tends to produce answers similar to old ones by biasing the search to look in areas that were previously successful – exploiting this effect gives our GA its learning behavior. When playing the game we periodically

inject a number of individuals from the case-base into the population, biasing our current search towards information from those individuals. Injection occurs by replacing the worst members of the population with individuals chosen from the case database through a "Probabilistic Closest to the Best" strategy [1]. Those individuals bring their "trap avoiding" knowledge into the population, increasing the likelihood of that knowledge being used in the final solution. Therefor increasing GAP's ability to avoid the trap.

6 Results

We present results showing

1. GAP can play the game effectively.
2. Replanning can effectively react to popups.
3. We can use case injection to learn to avoid the trap.

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

We first show that GAP can form efficient strategies. GAP is run against our test scenario 50 times, and we graph the min,max, and average population fitness against generation in Figure 5-Left. The graph shows a strong approach toward the optimum and in more the 95% of runs it gets within 5% of the optimum. Indicating that GAP can form effective strategies for playing the game.

To deal with opponent moves and the dynamic nature of the game we look at the effects of re-planning.

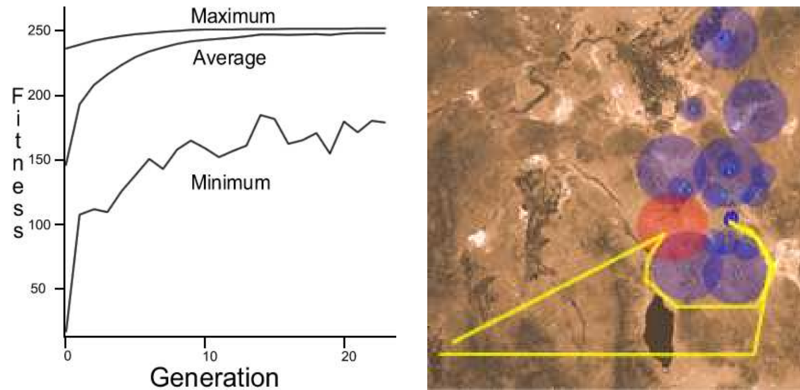

Fig. 5. Left: Best/Worst/Average Individual Fitness as a function of Generation - Averaged over 50 runs. Right: Final routes used during a mission involving replanning.

Figure 5-Right illustrates the effect of replanning by showing the final route followed inside a game. A yellow route was chosen, and when the popup occurred,

trapping the platforms, GAP redirected the strike force to retreat and attack from the rear. Replanning allows GAP to rebuild its routing information as well as modify its allocation to compensate for damaged platforms.

GAP's ability to learn to avoid the trap is shown in Figure 6. The figure compares the histograms of rc values produced by GAP with and without case injection. Case Injection leads to a strong shift in the kinds of rc's produced, biasing the population towards using green routes. The effect of this bias being a large and statistically significant increase in the frequency by which strategies containing green routes were produced ($2\% -> 42\%$). These results were based on 50 independent runs of the system and show that case injection does bias the search toward avoiding the trap.

Figure 7-left compares the fitnesses with and without case injection. Without case injection the search shows a strong approach toward the optimal yellow plan; with injection the population quickly converges toward the optimal green plan. Case injection applies a bias towards green routes, however the GA has a tendency to act in opposition of this bias, trying to search towards ever shorter routes. The ability of the GA to overcome the bias through manipulation of injected material is dependent on the size of the population and the number of generations it runs for, Figure 7-Right illustrates this effect. As the number of evaluations alloted to the GA is increased, the frequency of green routes being produced as a final solution decrease. Counteracting this tendency requires a careful balance of parameters to the GA and case-injection.

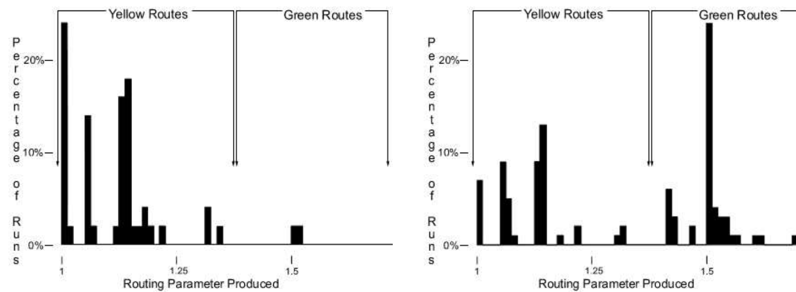

Fig. 6. Histogram of Routing Parameters produced without Case Injection.

7 Conclusions and Future Work

Results show that GAP is able to play the game, and that case injection can be used to to bias the search and incorporate information from outside sources not contained in the evaluation function. We had expected difficulty in biasing the search, but we had underestimated the GA's resilience towards searching away from the optimum. Our original expectations were to be able to apply a stronger

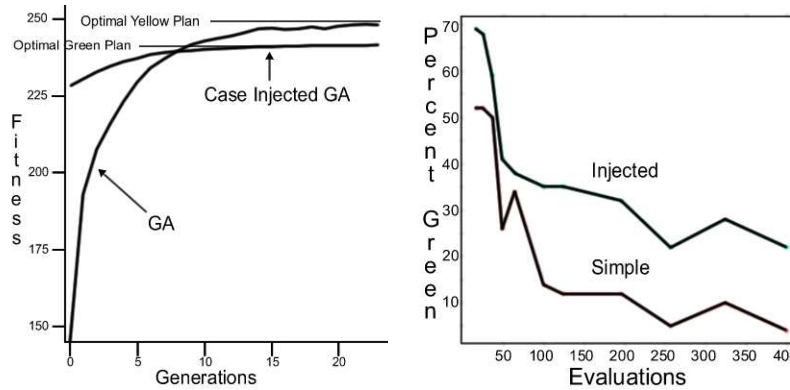

Fig. 7. Left: Effect of Case Injection on Fitness Inside the GA over time Right: Effect of Population Size and the Number of Generations on Percentage Green routes Produced

bias to the search, while 50% green is a significant improvement on 2% we had hoped for numbers in the range of 80 to 90%. Even after extensive testing with the parameters involved we were unable to bias the search towards consistently producing plans containing green routes. However, preliminary results from new work show that artificially inflating the fitness of individuals in the population that contain injected material is an effective way of maintaining this external information. This method appears to consistently produce green routes while maintaining an effective search across a range of problems without the need for parameter tuning.

There are a large number of interesting avenues in which to continue this research. Fitness inflation appears to solve one of our major problem in using case injection, further exploration of this technique is underway. We are also interested in capturing information from human players in order to better emulate their style of play. The game itself is also under major expansion, the next phase of research should involve a symmetric game involving aspects of resource management and much deeper strategies than those seen at the current level.

Acknowledgment

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., McDonnell, J.: Learning with case injected genetic algorithms. IEEE Transactions on Evolutionary Computation (To Appear in 2004)
2. Louis, S.J.: Evolutionary learning from experience. Journal of Engineering Optimization (To Appear in 2004)

3. Louis, S.J.: Genetic learning for combinational logic design. Journal of Soft Computing (To Appear in 2004)
4. Louis, S.J.: Learning from experience: Case injected genetic algorithm design of combinational logic circuits. In: Proceedings of the Fifth International Conference on Adaptive Computing in Design and Manufacturing, Springer-Verlag (2002) to appear
5. Louis, S.J., Johnson, J.: Solving similar problems using genetic algorithms and case-based memory. In: Proceedings of the Seventh International Conference on Genetic Algorithms, Morgan Kauffman, San Mateo, CA (1997) 283–290
6. 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
7. 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)
8. Yost, K.A.: A survey and description of usaf conventional munitions allocation models. Technical report, Office of Aerospace Studies, Kirtland AFB (Feb 1995)
9. 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
10. Fogel, D.B.: Blondie24: Playing at the Edge of AI. Morgan Kauffman (2001)
11. 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
12. 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
13. Kendall, G., Willdig, M.: An investigation of an adaptive poker player. In: Australian Joint Conference on Artificial Intelligence. (2001) 189–200
14. Samuel, A.L.: Some studies in machine learning using the game of checkers. IBM Journal of Research and Development **3** (1959) 210–229
15. Blizzard: Starcraft (1998, www.blizzard.com/starcraft)
16. Cavedog: Total annihilation (1997, www.cavedog.com/totala)
17. Inc., R.E.: Homeworld (1999, homeworld.sierra.com/hw)
18. Laird, J.E.: Research in human-level ai using computer games. Communications of the ACM **45** (2002) 32–35
19. Laird, J.E., van Lent, M.: The role of ai in computer game genres (2000)
20. Laird, J.E., van Lent, M.: Human-level ai's killer application: Interactive computer games (2000)
21. Tidhar, G., Heinze, C., Selvestrel, M.C.: Flying together: Modelling air mission teams. Applied Intelligence **8** (1998) 195–218
22. Serena, G.M.: The challenge of whole air mission modeling (1995)
23. 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)
24. Stout, B.: The basics of a* for path planning. In: Game Programming Gems, Charles River media (2000) 254–262