Evolving Galactic Conquerors

No Author Given

No Institute Given

Abstract. In spite of the increase of interest in applying computational intelligence techniques to computer games, there is one type that has not received so much attention: real-time strategy (RTS) games, where traditional artificial intelligence techniques fail to play at a human level because of the vast search spaces that they entail. We propose to make use of evolutionary algorithms fine-tune the behavior of a bot that play Planet Wars, the game that has been used for the Google Artificial Intelligence Challenge 2010. The behavior engine of the proposed bot is based on a set of rules established by means of exhaustive experimentation, followed by the application of an evolutionary algorithm to determine the constants needed by those rules. The resulting bot is able to beat the opponent (used to design it in the most challenging maps) and is well-placed (top 10%) in the Google AI challenge.

1 Introduction and Problem Description

A *Bot* in a computer game environment usually designs as an autonomous agent, which tries to compete in the game under the same conditions as a human player and cooperates with or fights/competes against a human player or other bots. Bots have been used extensively in First Person Shooter (FPS) games [1–4], where the human players fight in an scenario (or arena) against some of these bots, also known as Non-Playing Characters (NPCs).

Real-time strategy (RTS) games are those where the player owns some units (and/or structures) which he has to control, in order to beat the opponent, usually in a battle. In a typical RTS it is possible to create additional units and structures during the course of a game. This is generally limited by a requirement to expend accumulated resources. These are strategy-based games that usually work in real time, without any player stopping to wait for the results of the moves of the other player. Some examples of this type of game are the famous Command and Conquer™, Starcraft™, Warcraft™ or Age of Empires™ sagas.

RTS games are inherently difficult for its real-time nature (which is usually addressed by constraining the time that must be used to reach a decision) and also for the huge search space that is implicit in its action. This is probably one of the reasons why in the last Computational Intelligence in Games conference (IEEE-CIG 2010), just a few papers deal with this kind of games; the table of contents reveals just three papers, out of sixty, dealing with the subject. That is probably also one of the reasons why Google has chosen this kind of game for their Artificial Intelligence Challenge 2010.

The objective of the work presented in this paper is to implement the decision engine for a bot that plays a RTS game called *Planet Wars* or Galcon [5], which has been chosen for the Google AI Challenge 2010 [6]. This decision engine has been designed in two steps: first, a set of rules has been defined by means of exhaustive experimentation; they model the behavior of the bot, and depend on some parameters. The second step has been the application of a Genetic Algorithm (GA) [7] to evolve (and improve) these parameters off-line (not during a match, but previously to the game fights).

The rest of the paper is structured as follows: Section 2 describes the problem by presenting the Planet Wars game. Section 3 reviews related approaches to behavioral engine design in similar game-based problems. Section 4 presents the proposed method, termed GeneBot, detailing the finite state machine and the parameters which model its behavior, in addition to the GA which evolves them. The experiments and obtained results are described and analyzed in Section 5. Finally, the conclusions and future lines work are presented in Section 6.

2 The Planet Wars Game

The Planet Wars game chosen for the Google AI challenge (GAIC) [6] is an artificial intelligence competition where game-playing programs fight against others programs. As previously stated, our aim is to design the behavioral engine of a bot that play Planet Wars as well as possible. So it will try to win against any enemy (usually other bots).

Planet Wars is a game based on Galcon [5], but is designed to be simpler, since it is addressed to held bot's fights. The contest version of the game is for two players.

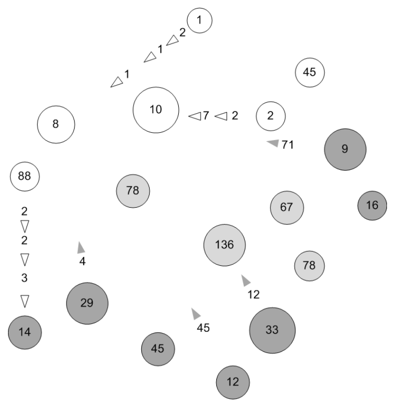

Fig. 1. Simulated screenshot of an early stage of a run in Planet Wars. White planets belongs to the player (blue color in the game), dark grey are planets are enemy's (red in the game), and light grey planets are nobody's. The triangles are fleets, and the numbers (in planets and triangles) mean starships.

A Planet Wars match takes place on a map which contains several planets, each of them with a number on it that represents the number of starships it hosts (see Figure 1). Each planet may have a different number of starships, and may belong to one of three different owners: the player, the enemy, or neutral (nobody). Ownership is shown with a colour, being blue for the player, red for the enemy, and grey for neutral (a non-playing character).

The objective is to defeat all of your opponent's planets. Even if it is a real-time game, the implementation is turn-based, with is a maximum number of turns to accomplish this objective. The player with more starships at the end of the match (set to two hundred turns in the challenge) wins, but if both players have the same number of forces when the game ends, it is a draw.

Each **planet** has some properties: *X and Y Coordinates*, *Owner's PlayerID*, *Number of Starships* and *Growth Rate* (on which depend the increasing of the number of starships in it). Players send fleets to conquer other planets (or to reinforce its own), and every **fleet** has the properties: *Owner's PlayerID*, *Number of Starships*, *Source PlanetID*, *Destination PlanetID*, *Total Trip Length*, and *Number of turns remaining until arrival*.

Each autonomous bot is implemented as a function that takes a list of planets and fleets, each one with the properties shown above, and outputs orders as a text file; in each turn, the player has to choose where to send fleets of starships, departing from any of his planets, to any other planet on the map. The fleets can take some turns to reach their destination. When a fleet reaches a planet, then it fights against the existent enemy's forces (losing one starship for each one at the planet) and, if it outnumbers the enemy units at the planet he/she becomes owner of that planet. If the planet already belongs to him, the incoming fleet is added as reinforcement. Each planet owned by a player (but not the "neutral" ones) will increase the forces there according to that planet's growth rate.

So, the problem to solve is the design of the function that considers the state of the map in each turn and decides the actions to perform in order to get and advantage over the enemy, and at the end, win the game.

There are two important constraints in the Google AI challenge: the first one is that a bot cannot keep any data from one turn to another (there is no memory). The second one is that the time limit to (decide and) perform the actions is just one second.

These restrictions make it difficult to implement an on-line metaheuristic approach, so our we run the evolutionary algorithm off-line and one a satisfactory state is found, it is sent to fight at the Challenge. Next we will see how other strategies for games have been designed.

3 State of the Art

In relation with the design or improvement of the Bots' AI in computer games, most attention has been directed at in FPS, starting with Doom™ or Quake™ [1] by the beginning of the 90s; more recently, the most important game (or environment) in which these studies have been developed is Unreal Tournament™ [4, 2, 3].

The behavior in these bots has been designed following different approaches: parameter-tuning based methods using reinforcement learning [8] of genetic algorithms to tune up the parameters of an expert system that controls the bots [9]. More complex approaches involve artificial Neural Networks (NNs) trained with reinforcement learning [10], or neural nets evolved for optimizing several objectives at the same time[11]. Some recent papers, such as [2], use both approaches described above at the same time: a genetic algorithm evolves the parameters of the Fuzzy Finite State Machine (FFSM) included in the game core that controls the bot's behavior, and a genetic programming to evolve rules as a replacement for the FFSM ones.

There are not so many papers devoted to RTS games, such as Age of Empires™ or Starcraft™; [12] is an example. However, in many RTS games, traditional artificial intelligence techniques fail to play at a human level because of the vast search spaces that they entail. In this sense, Ontano et at. [13] proposed to extract behavioral knowledge from expert demonstrations in form of individual cases. This knowledge could be reused via a case based behavior generator that proposed advanced behaviors to achieve specific goals. Other authors propose using coevolution for evolving team tactics [14]. However, the problem is how tactics are constrained and parametrized and how is the overall score computed.

On the other hand, RTS games usually have a 'static' or scripted (it does not change nor evolve) AI that controls the computer's actions. Once the user has learnt how such a game will react, the game quickly loses its appeal. In order to improve the users' gaming experience, Falke et al. [15] proposes a learning classifier system that can be used to equip the computer with dynamically-changing strategies that respond to the user's strategies, thus greatly extending the games playability.

Applying evolutionary algorithms to a parametrized tactic is relatively novel, and, as far as we know, nobody has applied it to this game, although the game forum mentions another challenger using genetic programming to evolve tactics. The idea in this work is to apply an evolutionary approach (mixed with some behavior model rules) to built the decision engine of a bot for playing in a 'simple' RTS game, getting advance of the best of both techniques.

4 GeneBot: The Galactic Conqueror

As previously stated in Section 2, the main constraint in each turn is the little processing time available to perform the correspondent actions (1 second). In addition, there is another key constraint, the complete absence of memory from one turn to another. These restrictions have strongly limited the design and implementation possibilities for our bot, since almost all the metaheuristics are based in a memory of solutions or the assignment of payoffs to previous actions in order to improve future behavior, and most of them are quite expensive in running time; running an evolutionary algorithm each turn, for instance, or a

Monte Carlo method is almost impossible. Besides, just the overall result of the strategy can be evaluated, without being able to optimize individual actions due to the lack of feedback from one turn to the next.

These are the reasons why we have decided to define a set of rules which models the on-line (during the game) bot's AI. These rules have been formulated through exhaustive experimentation, and strongly depend on some key parameter, which determines in the end the behavior of the bot. These parameters will be computed offline using an evolutionary algorithm, and fixed when the bot is sent off to fight to distant planets with other Google AI challengers. The number and nature of these parameters will be explained below.

This way, the main idea is to perform an off-line parameter optimization, by applying a Genetic Algorithm (GA) [7], so the resulting bot has been designed *GeneBot*. In this optimization step, our bot fights against a standard Google bot, which is included in the game kit that can be downloaded from the GAIC site, is quite simple but works very well in most of the game maps. It works as follows: for a state of the map, it seeks the planet that hosts most ships and the best planet to attack by calculating the ratio between the growth-rate and the number of ships. It attacks a single planet at a time, toggling between attack and non-attack modes; this one taken when it is under attack. Despite its simplicity it manages to win in enough maps if his opponent is not good enough. In fact the Google AI Contest recommends that any candidate bot should be able to always win the standard Google bot in order to have some kind of chance in the hall of fame; this is the baseline for even considering the bot as a challenger, and the number of turns it needs to win is an indicator of its quality.

From this standard bot, we derive another, which we call GAIBOT (Google AI Bot) whose strategy derives from this one, but with small variations. This bot works as follows: At the beginning of a turn, the bot tries to find a *base planet*, decided on the basis of a score whose weights will be evolved. The rest of the planets are named *colonies*. Then, it determines which *target planet* to attack (or to reinforce) in the next turns (since it can take some turns to get to that planet). If the planet to attack is neutral, the action is termed *expansion*; however, if the planet is owned by the enemy, the action is named *conquest*. The internal flow of the behavior of GeneBot with these states is shown in Figure 2.

The function considered to select the *target planet* is as follows:

$$Score(p) = \frac{p.NumStarships \cdot \omega_{NS-DIS} \cdot Dist(base, p)}{1 + p.GrowthRate \cdot \omega_{GR}} \tag{1}$$

where ω_{NS-DIS} and, ω_{GR} are weights related to the number of starships, the growth rate and the distance to the target planet. These values are adjusted using the GA, and will be commented in the next section. *base* is the best planet of the bot, and p is the planet to evaluate. The addition of '1' in the divisor ensures not to perform a division by zero.

Once the target enemy planet is designed, a particular colony can provide a part of its starships to the base planet. Moreover, if the distance between the colony and the target planet is less than the distance between the base and

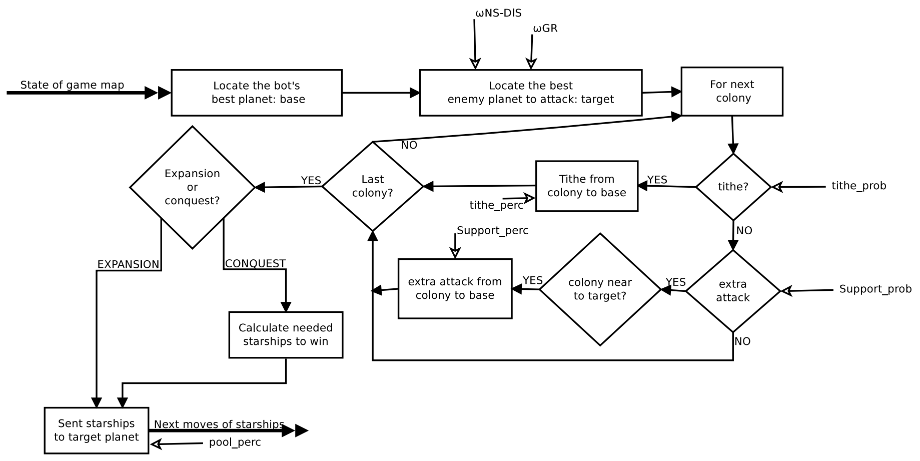

Fig. 2. GeneBot behavior finite state machine. It is shown where is considered each one of the parameters to evolve.

target planet, there is a likelihood that the colony also sent a number of troops to the target planet.

Once tithe and attack fleets from the colonies are scheduled, the remaining starships from the planet base are sent. If it is in expansion, the planet will not generate starships, so enough effectives to conquer the target planet will be sent with a certain number of extra units; this number is adjusted using the GA, and will be also commented in the next section. If it is trying to conquer a planet, the bot estimates the number of starships (grouped in fleets) needed to do it.

The following sections present the parameters which have been optimized using the GA, and the implemented algorithm to perform the optimization process. GAIBOT has intuitive and hand-fixed values for all weights mentioned; they are shown in Table 1. However, to evolve galactic conquerors we will leave all parameters as variables that will have to be optimized by the evolutionary algorithm. These parameters will be explained in the next section.

4.1 Parameters to Optimize

The set of parameters is composed by some weights, probabilities and amounts, that have been included in the rules that model the bot behavior (see them in the diagram in Figure 2). They have been determined following a systematic experimentation method, and have a strong influence in determining the final behavior of the bot:

- $tithe_{perc}$: percentage of starships the bot sends (regarding the number of starships in the planet).
- $tithe_{prob}$: probability that a colony sends a tithe to the base planet.

- ω_{NS-DIS}: weight of the number of starships hosted at the planet and the distance from the base planet to the target planet; it is used in the score function of target planet.
- ω_{GR}: weight of the planet growth rate in the target planet score function.
- $pool_{perc}$: proportion of extra starships that the bot sends from the base planet to the target planet.
- $support_{perc}$: percentage of extra starships that the bot sends from the colonies to the target planet.
- $support_{prob}$: probability of sending extra fleets from the colonies to the target planet.

Each parameter takes values in a different range, depending on its meaning, magnitude and significance in the game, but all of them are normalized to the $[0, 1]$ range.

The problem is to find the best parameter values in order to get the most suitable bot's behavior. Even finding the relative importance of each of them in the bot' AI modeling. To do this, a GA has been applied, as is commented in the next section.

4.2 The Genetic Algorithm

The implemented GA follows a *generational* [7] scheme, considering *elitism* to maintain the best solutions in time. Each individual (chromosome) is a floating point number array, with each component corresponding to one of parameters (see 4.1).

The genetic operators include a *BLX-alpha* crossover [16] (with alpha equal to 0.5) and a *gene mutator* which mutates the value of a random gene by adding or subtracting a random quantity in the $[0, 1]$ interval.

The *selection mechanism* implements a *2-tournament* [17]. The elitism has been implemented by replacing a random individual in the next population with the global best at the moment. The worst is not replaced in order to preserve diversity in the population.

The evaluation of one individual is performed by setting the correspondent values in the chromosome as the parameters for GeneBot's AI, and placing the bot inside a scenario to fight against a GAIBOT.

During the five matches, two values are calculated for the fitness evaluation: the first one is the number of turns that the bot has needed to win in each arena (WT), and the second is the number of games that the bot has lost (LT). Every generation bots are ranked considering the LT value; in case of coincidence, then the WT value is also considered as shown above: the best bot is the one that has won every single game; if two bots have the same WT value, the best is the one that needs less turns to win. This would probably be done better in an multiobjective way, to give the evolutionary algorithm a better chance of exploring the space; however, it is quite clear that the most important thing is to win the most games, or all in fact, and then minimize the number of turns; this way of ranking the population can be seen as an strategy of implementing a constrained optimization problem: minimize the number of turns needed to win *provided that* the individual is able to win every single game. Finally, in case of a complete draw (same value for LT and WT), zero is returned, meaning that no one has won.

5 Experiments and Results

To test the algorithm, different games have been played by pitting the standard bot (GAIBOT) against the optimized bot (GeneBot). The parameters considered in the GA are a population of 200 individuals, with a crossover probability of 0.8 (in a random point) and a mutation rate of 0.02. A two-individuals elitism has been implemented.

To evolve the behavior parameters, 30 runs of the GA have been performed, using 5 different (and representative) maps.

The evolutionary algorithm yields the values shown in Table 1.

	$tithe_{perc}$	$tithe_{prob}$	ω_{NS-DIS}	ω_{GR}	$pool_{perc}$	$support_{perc}$	$support_{prob}$
GAIBOT	0,1	0,5	1	1	0,25	0,5	0,9
GeneBot	0,294	0,0389	0,316	0,844	0,727	0,822	0,579

Table 1. Initial behavior parameters values of the original bot (GAIBOT), and the optimized values (evolved by a GA) for the best bot obtained using the evolutionary algorithm (GeneBot).

Results in Table 1 show that the best results are obtained by strategies where colonies have a low probability of sending tithe to the base planet (only 0.3), and those tithes send a few hosted starships, which probably implies that colonies should be left on its own to defend themselves instead of supplying the base planet. On the other hand, the probability for a planet to send starships to attack another planet is quite high (0.58), and the proportion of units sent is also elevated, showing that it is more important to attack with all the available starships than wait for reinforcements. Related to this property is the fact that, when attacking a target planet, the base one also sends a large number of extra starships (72.7 % of the hosted ships). Finally, to define the target planet to attack, the number of starships hosted in the planet is not as important as the growth range, but being the distance also an important value to take into account.

After this experiments, the value of the obtained parameters has been tested considering 100 different games (matches), where the 'evolved' GeneBot has fought against a standard GAIBOT. The results are shown in Table 2.

The number of turns a bot needs to win in a map is an important factor, since the bot should win. In the first turns, both bots handle the same number of starships, so making a difference in a few turns implies that the bot knows what to do and is able to accrue fastly many more ships (by conquering ship-growing planets). If it takes many turns, the bot actions have some room for improvement, and it would be even possible, if the enemy is a bit better than

	Turns			Victories
	Average & Std. Dev	Min	Max	
GAIBOT	210 ± 130	43	1001	99
GeneBot	159 ± 75	22	458	100

Table 2. Results after 100 games for standard bot (GAIBOT) and the best optimized bot (GeneBot) versus de Google Standard Bot. On average GeneBot needs less turns to win.

the one that Google uses as a baseline, to be defeated. By this reason, the number of turns is taken into account when determining the bot fitness: the faster it is able to beat the test-bot, the more likely it will defeat any enemy bot. Results show that the optimized bot (GeneBot) needs less turns on average to win.

6 Conclusions and Future Work

The Google AI Challenge 2010 is an international programming contest where game-playing programs (bots) fight against others in a RTS game called Planet Wars. This work presents a bot whose behavior parameters have been obtained using a Genetic Algorithm, and it has been shown that using this kind of algorithms increases the efficiency in playing versus hand-coded bots, winning more runs in a lower number of turns. Results obtained in this work show that it is important to attack planets with all available ships hosted in these planets, instead of storing this ships for future attacks.

At the close of this article, the contest is not over, so we can't know how our bot will finish. But in the Current Rankings (previous at the final challenge) the original bot was placed between 2000 and 2500. Right now, the best bot obtained with the GA is placed at TOP 1500.

As future work, a dynamic algorithm will be developed to modify the behavior parameters during runtime; for example, to improve the planet defenses when enemies are more aggressive, or vice-versa. Also, a deeper study with different types of AI bots and maps will be performed. If possible, we would use also other available bots to train. The baseline strategy will also have to be reassessed; since it is based in a certain sequence of events, it can only go as far as that strategy; even with the best parameters available, it could easily be defeated by other strategies. A more open approach to strategy design, even including genetic programming as mentioned by the other participants in the forum, could be quite interesting.

References

1. Laird, J.E.: Using a computer game to develop advanced AI. Computer (2001) 70–75
2. Esparcia-Alcázar, A.I., García, A.I.M., Mora, A., Guervós, J.J.M., García-Sánchez, P.: Controlling bots in a first person shooter game using genetic algorithms. In: IEEE Congress on Evolutionary Computation, IEEE (2010) 1–8

3. Mora, A.M., Ángel Moreno, M., Merelo, J.J., Castillo, P.A., Arenas, M.I.G., Laredo, J.L.J.: Evolving the cooperative behaviour in Unreal™ bots. In Yannakakis, G.N., Togelius, J., eds.: Computational Intelligence and Games, 2010. CIG 2010. IEEE Symposium On. (2010) 241–248
4. Small, R., Bates-Congdon, C.: Agent Smith: Towards an evolutionary rule-based agent for interactive dynamic games. In: Evolutionary Computation, 2009. CEC '09. IEEE Congress on. (2009) 660–666
5. Wikipedia: Galcon — Wikipedia, The Free Encyclopedia (2010) [Online; accessed 2-December-2010].
6. Google: Google AI Challenge 2010. http://ai-contest.com (2010)
7. Michalewicz, Z.: Genetic Algorithms + Data Structures = Evolution Programs. Third edn. Springer (1996)
8. McPartland, M., Gallagher, M.: Creating a multi-purpose first person shooter bot with reinforcement learning. In: Computational Intelligence and Games, 2008. CIG '08. IEEE Symposium On. (2008) 143–150
9. Cole, N., Louis, S., Miles, C.: Using a genetic algorithm to tune first-person shooter bots. In: Evolutionary Computation, 2004. CEC2004. Congress on. Volume 1. (2004) 139–145 Vol.1
10. Cho, B.H., Jung, S.H., Seong, Y.R., Oh, H.R.: Exploiting intelligence in fighting action games using neural networks. IEICE - Trans. Inf. Syst. **E89-D**(3) (2006) 1249–1256
11. Schrum, J., Miikkulainen, R.: Evolving multi-modal behavior in NPCs. In: Computational Intelligence and Games, 2009. CIG 2009. IEEE Symposium On. (2009) 325–332
12. Hong, J.H., Cho, S.B.: Evolving reactive NPCs for the real-time simulation game. In: Proceedings of the 2005 IEEE Symposium on Computational Intelligence and Games (CIG05). (2005)
13. Ontanon, S., Mishra, K., Sugandh, N., Ram, A.: Case-based planning and execution for real-time strategy games. In Weber, R., Richter, M., eds.: Case-Based Reasoning Research and Development. Volume 4626 of Lecture Notes in Computer Science. Springer Berlin / Heidelberg (2007) 164–178
14. Avery, P., Louis, S.: Coevolving team tactics for a real-time strategy game. In: Proceedings of the 2010 IEEE Congress on Evolutionary Computation. (2010)
15. Falke-II, W., Ross, P.: Dynamic Strategies in a Real-Time Strategy Game. E. Cantu-Paz et al. (Eds.): GECCO 2003, LNCS 2724, pp. 1920-1921, Springer-Verlag Berlin Heidelberg (2003)
16. Herrera, F., Lozano, M., Sánchez, A.M.: A taxonomy for the crossover operator for real-coded genetic algorithms: An experimental study. International Journal of Intelligent Systems **18** (2003) 309–338
17. Back, T., Fogel, D.B., Michalewicz, Z., eds.: Evolutionary Computation 1: Basic Algorithms and Operators. 1 edn. Taylor and Francis; 1st edition (2000)