Optimizing strategy parameters in a game bot*

A. Fernández-Ares, A.M. Mora, J.J. Merelo,
P. García-Sánchez and C. M. Fernandes

Depto. de Arquitectura y Tecnología de Computadores, U. of Granada
email: {antares,amorag,jmerelo,pgarcia,cfernandes}@geneura.ugr.es

Abstract. This paper proposes an Evolutionary Algorithm for fine-tuning the behavior of a bot designed for playing Planet Wars, a game that has been selected for the the Google Artificial Intelligence Challenge 2010. The behavior engine of the proposed bot is based on a set of rules established by means of heuristic experimentation, followed by the application of an evolutionary algorithm to set the constants, weights and probabilities needed by those rules. This bot eventually defeated the baseline bot used to design it in most maps, and eventually played in the Google AI competition, obtaining a ranking in the top 20%.

1 Introduction and Problem Description

In a computer game environment, a *Bot* is usually designed as an autonomous agent which tries to play under the same conditions as a human player, cooperating or competing with the human or with other bots.Real-time strategy (RTS) games is a sub-genre of strategy video games in which the contenders control units and structures, distributed in a playing area, 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, although usually restrained by a requirement to expend accumulated resources. These games, which include Starcraft™and Age of Empires™ , typically work in real time: the player does not wait for the results of other players' moves.

Google chose Planet Wars, a game of this kind that is a simplified version of the classic Galcon game (`http://galcon.com`), for their Artificial Intelligence Challenge 2010 (GAIC) (`http://ai-contest.com`), that pits user-submitted players against each other. The aim of this research is to design the behavioral engine of a bot that plays this game, trying to maximize its efficiency. 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 At a given time, each planet has a specific number of starships, and it may belong to the player, to the enemy, or it may be neutral (i.e., it belongs to nobody). Ownership

* Supported in part by Andalusian Government grant P08-TIC-03903, by the CEI BioTIC GENIL (CEB09-0010) Programa CEI del MICINN (PYR-2010-13) project, the Junta de Andalucía TIC-3903 and P08-TIC-03928 projects, and the Portuguese Fellowship SFRH /BPD / 66876 / 2009

is represented by a colour, being blue for the player, red for the enemy, and grey for neutral (a non-playing character). In addition, each planet has a growth rate that indicates how many starships are generated during each round of action and added to the starship fleet of the player that owns the planet.
The objective of the game is to conquer all of the opponent's planets. Although Planet Wars is a RTS game, the implementation has transformed it in a turn-based one, with each player having a maximum number of turns to accomplish the objective. The player with more starships at the end of the match (set to 200 actions in the challenge) wins. Each **planet** has some properties: *X and Y Coordinates*, *Owner's PlayerID*, *Number of Starships* and *Growth Rate*. Players send fleets to conquer other planets (or to reinforce its own), and every **fleet** also has a set of properties: *Owner's PlayerID*, *Number of Starships*, *Source PlanetID*, *Destination PlanetID*, *Total Trip Length*, and *Number of turns remaining until arrival*. A simulated turn is implemented by a second in duration. The bot only has this maximum time to order the next actions list. Moreover, a peculiarity of the problem is that the bot is unable to store any kind of knowledge about their actions in previous turns, the actions of his opponent or the game map, for instance. In short, every time elapses the simulation of a turn (second) the bot gets to meet again with a unknown map, like a new game. This inability to store knowledge about the gameplay makes the creation of the bot an interesting challenge.
In fact, each autonomous bot is implemented as a function that takes as an input the list of planets and fleets (the current status of the game), each one with its properties' values, and outputs a text file with the actions to perform. In each simulated shift, a player must choose where to send fleets of starships, departing from one of the player's planets and heading to other planet on the map. This is the only type of actions that the bot can do. The fleets can take some time steps to reach their destination. When a fleet reaches a planet, it fights against the existing enemy's forces (losing one starship for each one at the planet) and, in the case of outnumbering the enemy's units, the player becomes owner of that planet. If the planet already belongs to the player, 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. Therefore, the goal is to design/evolve a function that considers the state of the map in each simulated shift and decides the actions to perform in order to get an advantage over the enemy, and, at the end, win the game.
This paper proposes an evolutionary approach for generating the decision engine of a bot that plays *Planet Wars* (or Galcon), the RTS game that has been chosen for the Google AI Challenge 2010. This decision engine has been implemented in two steps: first, a set of rules, which, depending on some parameters, models the behavior of the bot, is defined by means of exhaustive experimentation; the second step applies a Genetic Algorithm (GA) to evolve (and improve) these parameters off-line, i.e., not during a match, but previously. Next we will present the state of the art in RTS games like this one.

2 State of the Art

RTS games show an emergent component [1] as a consequence of the two level AI (making decisions on the set of units, and one devoted to the each of these small units), since the units behave in many different (and sometimes unpredictable) ways. This feature can make a RTS game more entertaining for a player, and maybe more interesting for a researcher. In addition, 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. [2] 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.

So, recently a number of soft-computing techniques and algorithms, such as co-evolutionary algorithms [3] or multi-agent based methods [4], just to cite a few, have already been applied to handle these problems in the implementation of RTS games. For instance, there are many benefits attempting to build adaptive learning AI systems which may exist at multiple levels of the game hierarchy, and which co-evolve over time. In these cases, co-evolving strategies might be not only opponents but also partners operating at different levels [5] Other authors propose using co-evolution for evolving team tactics [6], but the problem is how tactics are constrained and parametrized and how compute the overall score.

Evolutionary algorithms have also been used in this field [7, 8], but they involve considerable computational cost and thus are not frequently used in online games. In fact, the most successful proposals correspond to EAs' off-line applications, that is, the EA works (for instance, to improve the operational rules that guide the bot's actions) while the game is not being played, and the results or improvements can be used later during the game. Through offline evolutionary learning, the quality of bots' intelligence can be improved, and this has been proved to be more effective than opponent-based scripts.

This way, in this work, an offline EA is applied to a parametrized tactic (set of behavior model rules) inside the Planet Wars game (a "simple" RTS game), in order to build the decision engine of a bot for that game, which will be considered later in the online matches. The process of designing this bot is presented next.

3 GeneBot: The Galactic Conqueror

As previously stated in Section 1, the main constraint in the environment is the limited processing time available to perform the correspondent actions (1 second). In addition, there is another key constraint: no memory is allowed, i.e., the bot cannot maintain a register of the results or efficiency of previous actions. These restrictions strongly limit the design and implementation possibilities for a bot, since many metaheuristics are based on a memory of solutions or on 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 in each time-step of 1 second, for instance, or a Monte Carlo method

[9], is almost impossible. Besides, only the overall result of the strategy can be evaluated. It is not possible 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. The rules have been formulated through exhaustive experimentation, and are strongly dependent on some key parameters, which ultimately determine the behavior of the bot. Anyway, there is only one type of action: move starships from one planet to another. The action is very simple, so the difficulty lies in choosing which planet creates a fleet to send forth, how many starships will be included in it and what will the target planet be.

The main example of this type of behavior is the Google-supplied baseline example, which we will call GoogleBot, included as a Java program in the game kit that can be downloaded from the GAIC site. GoogleBot works as follows: for a specific state of the map, the bot seeks for the planet owned by him that hosts the most ships and uses it as the base for the attack; The target will be chosen by calculating the ratio between the growth-rate and the number of ships for all enemy and neutral planets. Then it waits until the expeditionary attack fleet has reached its target; then it goes back to *attack mode*, selecting another planet as base for a new expedition.

Despite its simplicity, GoogleBot manages to win enough maps if its opponent is not good enough or is geared towards a particular situation or configuration. In fact the Google AI Contest recommends that any candidate bot should be able to win the GoogleBot every time in order to have any chance to get in the hall of fame; this is the baseline to consider the bot as a challenger, and the number of turns it needs to win is an indicator of its quality.

AresBot was designed to beat GoogleBot, and it works as follows: at the beginning of a turn, the bot tries to find its own *base planet*, decided on the basis of a score function. The rest of the planets are designed *colonies*. Then, it determines which *target planet* to attack (or to reinforce, if it already belongs to it) 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 designed *expansion*; however, if the planet is occupied by the enemy, the action is designed *conquest*. The base planet is also reinforced with starships coming from colonies; this action is called *tithe*, a kind of tax that is levied from the colonies to the imperial see. The rationale for this behavior is first to keep a stronghold that is difficult to conquer by the enemy, and at the same time to easily create a staging base for attacking the enemy. Furthermore, colonies that are closer to the target than to the base also send fleets to attack the target instead of reinforcing the base. This allows starships to travel directly to where they are required instead of accumulating at the base and then be sent. Besides, once a planet is being attacked it is marked so that it is not targeted for another attack until it is finished; this can be done straightforwardly since each attack fleet includes its target planet in its data.

The set of parameters is composed by weights, probabilities and amounts, that have been included in the rules that model the bot behavior. These pa-

rameters have been adjusted by hand, and they obviously totally determine the behavior of the bot. Its value and meaning are:

- $tithe_{perc}$ and $tithe_{prob}$: percentage of starships the bot sends (regarding the number of starships in the planet) and probability it happens.
- ω_{NS-DIS} and ω_{GR}: weight of the number of starships and planet growth rate 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.
- $pool_{perc}$ and $support_{perc}$: proportion and percentage of extra starships that the bot sends from the base planet 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. These values are used in expressions used by the bot to take decisions. For instance, the function considered to select the *target planet* is defined this way:

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

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. *base*, as explained above, is the planet with the maximum number of starships, and p is the planet to evaluate. The divisor is added 1 to protect against division by zero.

Once the target enemy planet is identified, a particular colony (chosen considering the tithe probability) 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 target planet, there is a likelihood that the colony also sent a number of troops to the target planet. When these movements are scheduled, a fleet is sent from the base planet with enough starships to beat the target.

All parameters in AresBot are estimated; however, they can be left as variable and optimized using an evolutionary algorithm [10] before sending out the bot to compete; we called the result *GeneBot*. The proposed GA uses floating point array to codify for all parameters shown in the previous versions, and follows a *generational* [10] scheme with *elitism* (the best solution always survives). The genetic operators include a *BLX-alpha* crossover [11] (with α 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. Each operator have an application rate (0.6 for crossover and 0.02 for mutator). These values were set by hand; since each run of the algorithm took a whole day.

The *selection mechanism* implements a *2-tournament*. Several other values were considered, but eventually the best results were obtained for this one, which represents the lowest selective pressure. 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 the diversity.

The evaluation of one individual is performed by setting the correspondent values in the chromosome as the parameters for GeneBot's behavior, and placing the bot inside a scenario to fight against a GoogleBot in five maps that were chosen for its significance.The bots then fights five matches (one in each map). The result of the match is not deterministic, but instead of doing several matches over each map, we consider that the different results obtained for a single individual in each generation will make only those that consistently obtain good results be kept within the population.

The performance of the bot is reflected in two values: 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. A multi-objective approach would in principle be possible here; however, it is 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.

4 Experiments and Results

To test the algorithm, different games have been played by pitting the standard bot (AresBot) and the optimized bot (GeneBot) versus the GoogleBot. The parameters considered in the GA are a population of 400 individuals, with a crossover probability of 0.6 (in a random point) and a mutation rate of 0.02. A 2 individuals elitism has been implemented.

The evaluation of each individual takes around 40 seconds, that is why we had time to make single run in time to enter the bot in the competition. In this run we obtained the values shown in Table 1.

	$tithe_{perc}$	$tithe_{prob}$	ω_{NS-DIS}	ω_{GR}	$pool_{perc}$	$support_{perc}$	$support_{prob}$
AresBot	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 (AresBot), 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 consider.

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

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

Table 2. Results after 100 games for our standard bot (AresBot) and the best optimized bot (GeneBot) versus the Google Standard Bot.

The number of turns a bot needs to win in a map is the most important factor of the two considered in the fitness, since it needs to beat GoogleBot in all maps for having any kind of chance in the challenge. In the first turns, the two bots handle the same number of starships so making a difference in a few turns implies the bot knows what to do and is able to accrue many more ships (by conquering ship-growing planets) fast. If it takes many turns, the actions of the bot have some room for improvement, and it would be even possible, if the enemy is a bit better than the one Google issues as a baseline, to be defeated.

In general, the improvement to the original AresBot offered by the algorithm could seem small from the purely numeric point of view; GeneBot is able to win in one of the maps where AresBot was beaten, which was one of the 5 selected to perform evolution, and the aggregate number of generations is around 10%. However, this small advantage confers some leverage to win more battles, which in turn will increase its ranking in the Google AI challenge. This indicates that an evolutionary algorithm holds a lot of promise in optimizing any kind of behavior, even a parametrized behavior like the one programmed in GeneBot. However, a lot of work remains to be done, either to compete in next year's challenge, or to explore all the possibilities the genetic evolution of bot behavior can offer.

5 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. In this paper we wanted to show how evolutionary algorithms can be applied to obtain good results in a real-world challenge, by submitting to the competition a bot whose behavioral parameters are 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 these ships for future attacks.

The bot described here eventually finished 1454[1], winning nine matches and losing six; this placed it among the 32% best, which means that at least this technique of fine-tuning strategy parameters shows a lot of promise; however, it can only take you as far as the strategy allows. This was an improvement of more than 1000 positions over the non-optimized version.

In the future we will try to improve the baseline strategy, and even make the evolutionary process choose between different possible strategies; we will also try to make evolution faster so that we can try different parametrizations to obtain bots as efficient as possible.

References

1. Sweetser, P.: Emergence in Games. Game development. Charles River Media, Boston, Massachussetts (2008)
2. 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
3. Keaveney, D., OŔiordan, C.: Evolving robust strategies for an abstract real-time strategy game. In Press, I., ed.: International Symposium on Computational Intelligence in Games, Milano. Italy (2009) 371–378
4. Hagelbäck, J., Johansson, S.J.: A multiagent potential field-based bot for real-time strategy games. Int. J. Comput. Games Technol. **2009** (2009) 4:1–4:10
5. Livingstone, D.: Coevolution in hierarchical ai for strategy games. In: IEEE Symposium on Computational Intelligence and Games (CIG05), Essex University, Colchester, Essex, UK, IEEE (2005) 190–194
6. Avery, P., Louis, S.: Coevolving team tactics for a real-time strategy game. In: Proceedings of the 2010 IEEE Congress on Evolutionary Computation. (2010)
7. Ponsen, M., Munoz-Avila, H., Spronck, P., Aha, D.W.: Automatically generating game tactics through evolutionary learning. AI Magazine **27**(3) (2006) 75–84
8. Jang, S.H., Yoon, J.W., Cho, S.B.: Optimal strategy selection of non-player character on real time strategy game using a speciated evolutionary algorithm. In: Proceedings of the 5th IEEE Symposium on Computational Intelligence and Games (CIG'09), Piscataway, NJ, USA, IEEE Press (2009) 75–79
9. Lucas, S.: Computational intelligence and games: Challenges and opportunities. International Journal of Automation and Computing **5**(1) (2008) 45–57
10. Michalewicz, Z.: Genetic Algorithms + Data Structures = Evolution Programs. Third edn. Springer (1996)
11. 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

[1] Final ranking at: `http://ai-contest.com/profile.php?user_id=8220`