Optimizing player behavior in a real-time strategy game using evolutionary algorithms

A. Fernández-Ares, A.M. Mora, J.J. Merelo, P. García-Sánchez and C. Fernandes
Departamento de Arquitectura y Tecnología de Computadores
University of Granada
Email: {antares,amorag,jmerelo,pgarcia,cfernandes}@geneura.ugr.es

***Abstract*— This paper describes an Evolutionary Algorithm for evolving the decision engine of a bot designed to play the Planet Wars game. This game, which has been chosen for the Google Artificial Intelligence Challenge in 2010, requires that the artificial player is able to deal with multiple objectives, while achieving a certain degree of adaptability in order to defeat different opponents in different scenarios. The decision engine of the bot is based on a set of rules that have been defined after an empirical study. Then, an Evolutionary Algorithm is used for tuning the set of constants, weights and probabilities that define the rules, and, therefore, the global behavior of the bot. The paper describes the Evolutionary Algorithm and the results attained by the decision engine when competing with other bots. The proposed bot defeated a baseline bot in most of the playing environments and obtained a ranking position in top-20% of the Google Artificial Intelligence competition.**

I. Introduction and Problem Description

One of the most important elements in computer games are the *bots*. Bots are autonomous agents which compete or cooperate with the human player to increase the challenge of a game, being the intelligence of the bot one fundamental parameter in video-games design [1]. The bots are mainly used in First Person Shooter (FPS) games [2], [3], [4], where these agents are the enemies of the human player.

Real-time strategy (RTS) games form 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 typically work in real time: the player does not wait for the results of other players' moves. Some examples of this type of games are the famous Command and Conquer™, Starcraft™, Warcraft™ and Age of Empires™ sagas.

RTS games often employ two "levels" of AI [5]: one represented by a NPC, making decisions on the set of units (workers, soldiers, machines, vehicles or even buildings), and another one devoted to the each of these small units. These games are inherently difficult for their real-time nature (which is usually addressed by constraining the time that can be used to reach a decision) and also for the huge search space that is implicit in its action. This is probably why, in the last Computational Intelligence in Games conference (IEEE-CIG 2010), less than 10% of the papers deal with this kind of games. Such difficulties are probably also one of the reasons why Google has chosen this kind of game for their Artificial Intelligence Challenge 2010. In this challenge, real time is sliced into one second *turns*, with players receiving the chance to play sequentially; however, *actions* happen at the *simulated* same time, thus becoming a trait of the particular implementation and not a feature of the game itself.

This paper proposes an evolutionary approach for generating the decision engine of a bot that plays *Planet Wars* (or Galcon [6]), the RTS game that has been chosen for the Google AI Challenge 2010 [7]. 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) [8] to evolve (and improve) these parameters off-line, i.e., not during a match, but prior to the game fights.

The parameter tuning previously an algorithm run (or bot in this case) is a wide research area [9]. A bot with good parameter values can be orders of magnitude better than one with poorly chosen ones. So, this kind of technique is an optimization problem itself, moreover it can be also used to obtain information about the bot to tune, which parameters are more important or bot's behavioural data during the games.

The paper is structured as follows: Section II addresses the problem by describing the Planet Wars game. Section III reviews related approaches to behavioral engine design in similar game-based problems. Section IV presents the proposed method, termed GeneBot, detailing the finite state machine and the parameters which model its behavior, in addition to the GA used to evolves the strategies. The experiments and results are described and discussed in Section V. Finally, the conclusions and future lines of research are presented in Section VI.

II. The Planet Wars Game

The Google AI challenge (GAIC) [7] is an AI competition where the participants create programs (bots) to compete against other. The game chosen for the competition, Planet Wars, has been studied in this paper to design the behavioral engine of a bot which plays it, trying to maximize its efficiency. Planet Wars is a simplified version of Galcon [6], since it is aimed to perform bot's fights. The contest version of the game is for two players.

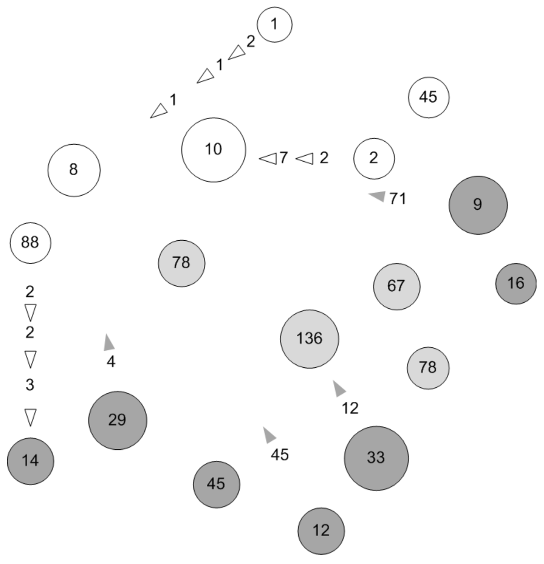


Fig. 1. Simulated screenshot of an early stage of a run in Planet Wars. White planets belongs to the player (blue color in the game), dark grey belong to the opponent (red in the game), and light grey planets belong to no player. The triangles are fleets, and the numbers (in planets and triangles) represent the 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). At a given time step, 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 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 defeat 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 its actions in previous turns, the actions of its 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 is allowed to perform. 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 the owner of that planet. If the planet already belongs to the player, the incoming fleet is added as reinforcement. In each time-step, the forces in each planet owned by a player (but not the "neutral" ones) are increased 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.

The commented constraints in the Google AI challenge (a bot cannot keep any data from one turn to another, and the one second time limit) make it difficult to implement an on-line metaheuristic approach. Therefore, the proposed Evolutionary Algorithm (EA), a Genetic Algorithm (GA), is executed off-line and once a satisfactory state is found, the solution (bot) is sent to fight at the Challenge.

III. State of the Art

Videogames have become one of the greatest sectors in the leisure industry, which cost in development sometimes higher than blockbuster movies. This cost is mainly devoted to increase the graphical quality of the games, as a measure to ensure a best-seller product, instead innovate in challenging the player skills. Nowadays, computers have a higher processing power and the user is hardly astonished by the graphical component of a game. So, the players have turned their attention to other aspects of the game. In particular, they mostly request opponents exhibiting intelligent behavior, or just better human-like behaviors [10].

During these years, the games AI researching have followed a different path, mainly starting with the improvement of FPS Bot's AI with Doom™ or Quake™ [1] by the beginning of the 90s; and the most famous environment inside this kind of games, Unreal Tournament ™ [4], [2], [3].

Most of the researches have been done on relatively simple games such as Super Mario [11], Pac-Man [12] or Car Racing Games [13], being many competitions devoted to choose the best bot playing in each of them.

RTS games show an emergent component [14] as a consequence of the cited two level AI, 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. There are many research problems with regard to the AI for RTSs, including planning in an uncertain world with incomplete information; learning; opponent modelling and spatial and temporal reasoning [15], [16].

However, the reality in the industry is that in most of the RTS games, the NPC (bot) is basically controlled by a fixed script that has been previously programmed (following a finite state machines or a decision tree, for instance). 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, some authors such as Falke et al. [17] proposed 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.

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. [18] 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 [19], [20], [21] or multi-agent based methods [22], 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 [23] Other authors propose using co-evolution for evolving team tactics [24]. However, the problem is how tactics are constrained and parametrized and how is the overall score computed.

Evolutionary algorithms have also been used in this field [25], [26], but they involve considerable computational cost and thus are not frequently used in on-line games. In fact, the most successful proposals for using EAs in games correspond to off-line applications [27], 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 in commercial games can be improved, and this has been proven to be more effective than opponent-based scripts.

This way, in this work, an offline GA is applied to a parameterized 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.

IV. GeneBot: The Galactic Conqueror

In section II we explained that the main constraint in the environment is the limited processing time available to perform the correspondent actions (1 second). In addition, another important constraint stated that no memory is allowed, that is, 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 [28], 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.

Those reasons have meant the definition of 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; but the nature of this movement will be different depending on whether the target planet belongs to oneself or the enemy. As the action itself is very simple, 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; the behavior of this bot will be explained next, followed by our first attempt at defeating this bot, which we called AresBot (described in subsection IV-B. Finally, we will explain the main mechanisms governing the bot proposed in this paper, called GeneBot IV-C.

A. GoogleBot, a basic bot for playing Planet Wars

The development kit of GAIC includes an example of bot, called GoogleBot. This bot is quite simple, but is designed to work well independently of the map configuration, so it is able to defeat bots that are optimized for a particular kind of map, for instance, those configured to work well for situations in which enemy bases are far away (or the other way round).

GoogleBot works as follows: for a specific state of the map, the bot seeks for the planet it owns that hosts most of the 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. It waits until the expeditionary attack fleet has reached its target. When it lands, it goes back to *attack mode*, selecting another planet as base for a new expedition.

In spite of its simplicity, the Google bot 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.

Next we will show our first initial design for a decent challenger for GoogleBot.

B. Operation AresBot

As previously said, GoogleBot has a simple behaviour, since there is no movement of troops from one planet to another that might need it more, and also attacks are staged from a single planet at the time, that is, if it owns two planets with the same amount of troops, or a difference of one, it will not attack two planets. At any particular moment, besides, it is only or attacking or waiting, making for a not very efficient use of time.

So the researching started trying to face the Google AI challenge by designing a new bot that is, at least, able to offer a hand-coded strategy that is better than the one scripted in the GoogleBot. AresBot 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 structure.

The internal flow of GAIBot's behavior with these states is shown in Figure 2.

The set of parameters is composed by weights, probabilities and amounts, that have been included in the rules that model the bot behavior (shown in the diagram in Figure 2). These parameters have been adjusted by hand, and they obviously totally determine the behavior of the bot. Its value and meaning are:

- $tithe_{perc}$: 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. 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 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.

Once tithe and attack fleets from the colonies are scheduled, the remaining starships from the base planet are sent to attack the target. If it is in an expansion mode, the planet will not generate starships, so enough troops to conquer the target planet will be sent with a certain number of extra units, since the neutral planets do not increase its number of starships. On the other hand, if it is trying to conquer a planet, the bot estimates the number of starships (grouped in fleets) needed to do it.

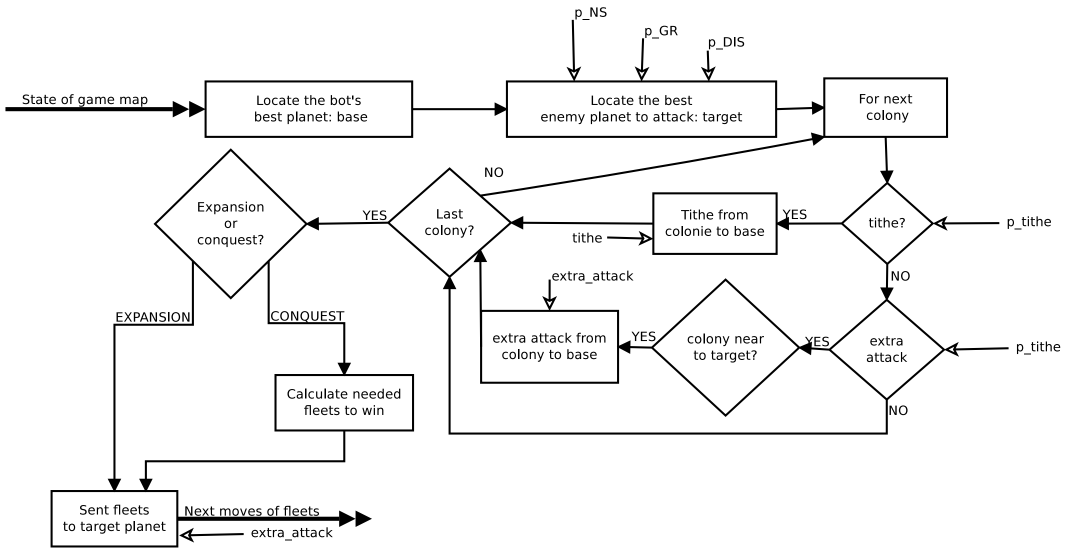


Fig. 2. Diagram of states governing the behavior of AresBot and GeneBot, with the parameters that will be evolved highlighted. These parameters are set by hand in AresBot, and evolved for GeneBot.

This bot already had a behavior more complex than GoogleBot, and was able to beat it in 99 out of 100 maps; however, it needed lots of turns to beat it; this meant that faster bots or those that developed an strategy quite fast would be able to beat it quite easily. That is why we decided to perform a systematic exploration of the values for the parameters shown above, in order to find a bot that is able to compete successfully (to a certain point) in the google AI challenge.

C. GeneBot: a genetically optimized AresBot

This way, the main idea is to perform an offline parameter optimization, by applying a Genetic Algorithm (GA) to the set of parameters that rule AresBot (explained in subsection IV-B), so that the resulting bot, called *GeneBot*, is the result of an optimization process. The evolutionary algorithm is not part of the competing bot; once the parameters have been found, they are fixed and the bot uploaded to the Google AI Challenge site behaved just like other hand-coded bot, except that the constants, probabilities and other numbers were found using an evolutionary algorithm.

Therefore, the objective is to find the parameter values that maximize the efficiency of the bot's behavior, and study the relative importance of each of them in the bot' AI modelling.

The proposed GA uses floating point array to codify for all parameters shown in the previous versions, and follows a *generational* [8] scheme with *elitism* (the best solution always survives). The genetic operators include a *BLX-alpha* crossover [29] (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 there was no time for finding the best value for them.

The *selection mechanism* implements a *2-tournament* [30], where every two individuals compete for being chosen as one of the parents of the next population. Some other mechanisms were considered (such as roulette wheel), 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 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 behavior, and placing the bot inside a scenario to fight against a GoogleBot in five maps that were chosen for its significance. These maps are as follows:

- Bases towards the middle of the map, and the best targets between them. This map is shown in figure 3.

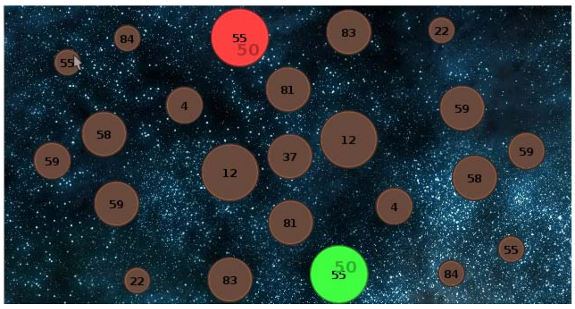


Fig. 3. One of the maps used to *train* the bots. The bot planet is shown in green, bottom center, with the number 55; the enemy planet is shown in red, has the same number as a label and is placed at the top center.

- Very few and widely spread planets, bases away from each other
- Bases apart from each other, planets away from bases in the *corners* of the map.

- Bases as far apart as possible, with planets crowding the center of the map
- The last map is similar to the first, but planets are in the corners of the map instead of the center.

These maps represent a wide range of situations, and it was considered that if a bot was able to beat GoogleBot in all five, and also in a minimum amount of turns, it would have a high probability of succeeding at least in number of games.

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, meaning that no one has won.

The source code of all these bots can be found at: https://forja.rediris.es/svn/geneura/Google-Ai2010.

V. Experiments and Results

In order to test the GA, several games have been played by contending the standard bot (AresBot) and the optimized bot (GeneBot) against the GoogleBot. The (heuristically found) experimental parameters can be seen in Table I.

Number of individuals	400
Mutation probability	0.02
Crossover probability	0.6
Replacement policy	2-individuals elitism

TABLE I
Parameters used in the Genetic Algorithm.

The evaluation of each individual takes around 40 seconds, that is why we had time to make single run of the evolutionary algorithm during a day and a half, in time to enter the bot in the competition. The evolution of the number of turns the best bot needed to win (one of the two, and the most important component of the fitness) is shown in Figure 4 shows that as the evolution progresses, the number of turns needed to win in five maps decreases.

The evolutionary algorithm yields the values shown in Table II.

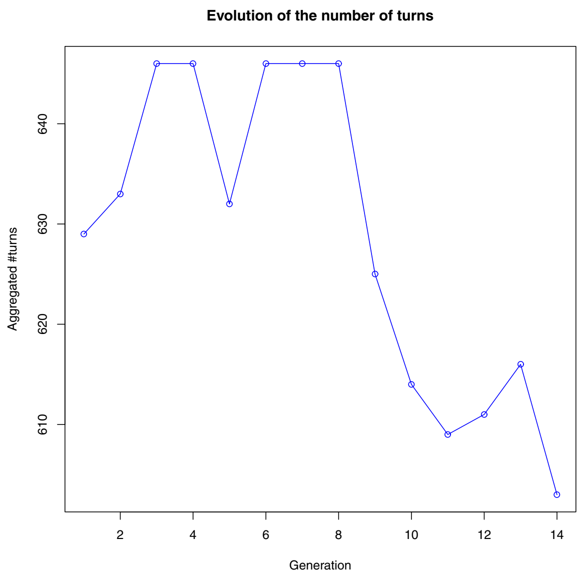


Fig. 4. Best bot between generations: as the evolution progresses (number of generations increases), the aggregate number of turns needed to win 5 maps decreases on average; however, since the result of combat, and thus the fitness, is not totally deterministic, it can increase from one generation to the next.

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

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. This is why the number of turns is considered when assigning the fitness to the bot; the faster it is able to

	$tithe_{perc}$	$tithe_{prob}$	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 II
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).

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

TABLE III
RESULTS AFTER 100 GAMES FOR OUR STANDARD BOT (ARESBOT) AND THE BEST OPTIMIZED BOT (GENEBOT) VERSUS THE GOOGLE STANDARD BOT.

beat the test bot, the better chance it will have to defeat any enemy bot. In addition, the number of turns to win each of the matches (for both bots) are presented in Figure 5. As far as GeneBot is concerned, this figure shows that the optimized bot needs fewer turns than AresBot, its non-optimized version, in average to win.

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%. Hoever, 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.

VI. 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 paper intends to show how Evolutionary Algorithms (EAs) can be applied, obtaining good results, to a real-world challenge, by submitting to the competition a bot whose behavioral parameters are tuned by a Genetic Algorithm. It is demonstrated that EAs can improve the efficiency of a hand-coded bot, winning more runs in a lower number of turns. Results show that is important to attack planets with the available ships, instead of storing those shops for future attacks.

The described bot finished 1454 in the contest, winning nine matches and losing seven, which placed it amongst the top-20% bots, meaning that the evolutionary approach for fine-tuning the parameters is at least promising for these types of problems. Although the approach is limited by the strategy itself, it is a good improvement over the original AresBot which started in a position below 2000 (to be later replaced by this version, since only one submission was admitted).

As future work, we intend to develop a dynamic algorithm for modifying the parameters on-line (for instance, to improve the planet's defenses when enemies are more aggressive, or vice-versa). In addition, a deeper study with different types of AI bots and maps will be performed. If possible, other available bots will be used for training. The baseline strategy will also have to be reassessed. Since it is based on 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, is a promising approach.

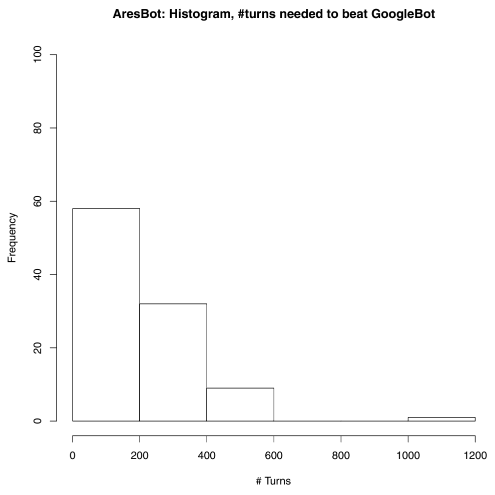
In the evolutionary algorithm front, several improvements might be attempted. For the time being, the bot is optimized against a single opponent; instead, several opponents might be tried (the three described in these papers, for instance), or even other individuals from the same population, in a coevolutionary approach. Another option will be to change the bot from a single optimized strategy to a set of strategies and rules that can be chosen also using an evolutionary algorithm. Finally, a multi-objective EA will be able to explore the search space more efficiently, although in fact the most important factor is the overall number of turns needed to win.

ACKNOWLEDGEMENTS

This work has been supported in part by the Junta de Andalucía TIC-3903 project, and by the project FCT (ISR/IST plurianual funding) through the PIDDAC Program funds. C. Fernandes wishes to thank FCT, Ministério da Ciência e Tecnologia, his Research Fellowship SFRH/BPD/66876/2009.

REFERENCES

[1] J. E. Laird, "Using a computer game to develop advanced AI," *Computer*, pp. 70–75, 2001.



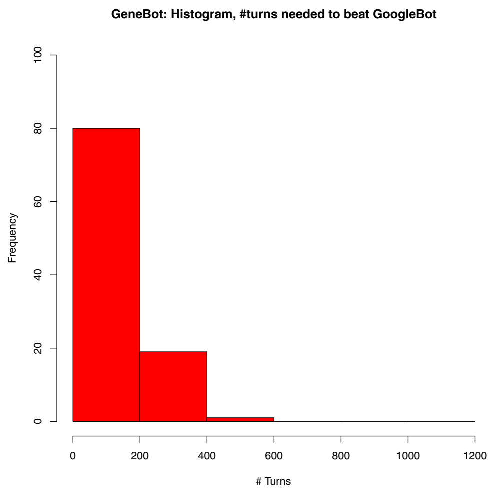


Fig. 5. Histogram of the number of turns needed to win, measured for the 100 examples maps in the GAIC kit. AresBot (top) needs, on average, more turns to win than GeneBot (bottom).

[2] A. I. Esparcia-Alcázar, A. I. M. García, A. Mora, J. J. M. Guervós, and P. García-Sánchez, "Controlling bots in a first person shooter game using genetic algorithms," in *IEEE Congress on Evolutionary Computation*. IEEE, 2010, pp. 1–8.

[3] A. M. Mora, M. Ángel Moreno, J. J. Merelo, P. A. Castillo, M. I. G. Arenas, and J. L. J. Laredo, "Evolving the cooperative behaviour in Unreal™ bots," in *Computational Intelligence and Games, 2010. CIG 2010. IEEE Symposium On*, G. N. Yannakakis and J. Togelius, Eds., August 2010, pp. 241–248.

[4] R. Small and C. Bates-Congdon, "Agent Smith: Towards an evolutionary rule-based agent for interactive dynamic games," in *Evolutionary Computation, 2009. CEC '09. IEEE Congress on*, May 2009, pp. 660–666.

[5] J. B. Ahlquist and J. Novak, *Game Artificial Intelligence*, ser. Game Development Essentials. Canada: Thompson Learning, 2008.

[6] Wikipedia, "Galcon — Wikipedia, The Free Encyclopedia," 2010, [Online; accessed 2-December-2010]. [Online]. Available: http://en.wikipedia.org/w/index.php?title=Galcon&oldid=399245028

[7] Google, "Google AI Challenge 2010," *http://ai-contest.com*, 2010.

[8] Z. Michalewicz, *Genetic Algorithms + Data Structures = Evolution Programs*, 3rd ed. Springer, 1996.

[9] A. Eiben and S. Smit, "Parameter tuning for configuring and analyzing evolutionary algorithms," *Swarm and Evolutionary Computation*, vol. In Press, Corrected Proof, pp. –, 2011.

[10] L. Lidén, "Artificial stupidity: The art of intentional mistakes," in *AI Game Programming Wisdom 2*. Charles River Media, INC., 2004, pp. 41–48.

[11] J. Togelius, S. Karakovskiy, J. Koutnik, and J. Schmidhuber, "Super mario evolution," in *Proceedings of the 5th IEEE Symposium on Computational Intelligence and Games (CIG'09)*. Piscataway, NJ, USA: IEEE Press, 2009.

[12] E. Martín, M. Martínez, G. Recio, and Y. Saez, "Pac-mant: Optimization based on ant colonies applied to developing an agent for ms. pacman," in *Computational Intelligence and Games, 2010. CIG 2010. IEEE Symposium On*, G. N. Yannakakis and J. Togelius, Eds., August 2010, pp. 458–464.

[13] E. Onieva, D. A. Pelta, J. Alonso, V. Milanés, and J. Pérez, "A modular parametric architecture for the torcs racing engine," in *Proceedings of the 5th IEEE Symposium on Computational Intelligence and Games (CIG'09)*. Piscataway, NJ, USA: IEEE Press, 2009, pp. 256–262.

[14] P. Sweetser, *Emergence in Games*, ser. Game development. Boston, Massachussetts: Charles River Media, 2008.

[15] M. Buro, "Call for AI research in RTS games," in *AAAI workshop on Challenges in Game AI*, D. Fu and J. Orkin, Eds., San Jose, July 2004, pp. 139–141.

[16] J.-H. Hong and S.-B. Cho, "Evolving reactive NPCs for the real-time simulation game," in *Proceedings of the 2005 IEEE Symposium on Computational Intelligence and Games (CIG05)*, 4-6 April 2005.

[17] W. Falke-II and P. Ross, "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.

[18] S. Ontanon, K. Mishra, N. Sugandh, and A. Ram, "Case-based planning and execution for real-time strategy games," in *Case-Based Reasoning Research and Development*, ser. Lecture Notes in Computer Science, R. Weber and M. Richter, Eds. Springer Berlin / Heidelberg, 2007, vol. 4626, pp. 164–178.

[19] C. Miles and S. J. Louis, "Co-evolving real-time strategy game playing influence map trees with genetic algorithms," in *International Congress on Evolutionary Computation*, I. Press, Ed., Portland, Oregon, 2006.

[20] N. Beume *et al.*, "Intelligent anti-grouping in real-time strategy games," in *International Symposium on Computational Intelligence in Games*, I. Press, Ed., Perth, Australia, 2008, pp. 63–70.

[21] D. Keaveney and C. ORiordan, "Evolving robust strategies for an abstract real-time strategy game," in *International Symposium on Computational Intelligence in Games*, I. Press, Ed., Milano. Italy, 2009, pp. 371–378.

[22] J. Hagelbäck and S. J. Johansson, "A multi-agent potential field-based bot for a full RTS game scenario," in *AIIDE*, C. Darken and G. M. Youngblood, Eds. The AAAI Press, 2009.

[23] D. Livingstone, "Coevolution in hierarchical ai for strategy games," in *IEEE Symposium on Computational Intelligence and Games (CIG05)*. Essex University, Colchester, Essex, UK: IEEE, 2005.

[24] P. Avery and S. Louis, "Coevolving team tactics for a real-time strategy game," in *Proceedings of the 2010 IEEE Congress on Evolutionary Computation*, 2010.

[25] M. Ponsen, H. Munoz-Avila, P. Spronck, and D. W. Aha, "Automatically generating game tactics through evolutionary learning," *AI Magazine*, vol. 27, no. 3, pp. 75–84, 2006.

[26] S.-H. Jang, J.-W. Yoon, and S.-B. Cho, "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, pp. 75–79.

[27] P. Spronck, I. Sprinkhuizen-Kuyper, and E. Postma, "Improving opponent intelligence through offline evolutionary learning," *International Journal of Intelligent Games & Simulation*, vol. 2, no. 1, pp. 20–27, February 2003.

[28] S. Lucas, "Computational intelligence and games: Challenges and opportunities," *International Journal of Automation and Computing*, vol. 5, no. 1, pp. 45–57, 2008.

[29] F. Herrera, M. Lozano, and A. M. Sánchez, "A taxonomy for the crossover operator for real-coded genetic algorithms: An experimental study," *International Journal of Intelligent Systems*, vol. 18, pp. 309–338, 2003.

[30] T. Back, D. B. Fogel, and Z. Michalewicz, Eds., *Evolutionary Computation 1: Basic Algorithms and Operators*, 1st ed. Taylor and Francis; 1st edition, 2000.