Evolving the strategies of agents for the ANTS game

J. Carpio[1], P. García-Sánchez[2], A.M. Mora[2], J.J. Merelo[2],
J. Caraballo[1], F. Vaz[1], C. Cotta[3]

[1] Dept. of Computer Science, University of Huelva, Spain
[2] Dept. of Computer Architecture and Technology, University of Granada, Spain
[3] Dept. of Computer Languages and Computer Sciences, Málaga, Spain
jose.carpio@dti.uhu.es

Abstract. This work studies the performance and the results of the application of Evolutionary Algorithms (EAs) for evolving the decision engine of a program, called in this context *agent*, which controls the player's behaviour in an real-time strategy game (RTS). This game was chosen for the Google Artificial Intelligence Challenge in 2011, and simulates battles between teams of ants in different types of maps or mazes. According to the championship rules the agents cannot save information from one game to the next, which makes impossible to implement an EA 'inside' the agent, i.e. on game time (or on-line), that is why in this paper we have evolved this engine off-line by means of an EA, used for tuning a set of constants, weights and probabilities which direct the rules. This evolved agent has fought against other successful bots which finished in higher positions in the competition final rank. The results show that, although the best agents are difficult to beat, our simple agent tuned with an EA can outperform agents which have finished 1000 positions above the untrained version.

1 Introduction

Real-Time Strategy (RTS) games are a sub-genre of strategy-based videogames in which the contenders control a set of units and structures that are distributed in a playing arena. The game objective is normally eliminating all the enemy units. It is usually possible to create additional units and structures during the course of the game, at a cost in resources. Another usual feature is their real time nature, so the player is not required to wait for the results of other players' moves as in turn-based games. Starcraft$^{\text{TM}}$, Warcraft$^{\text{TM}}$ and Age of Empires$^{\text{TM}}$ are some examples of RTS games.

The 2011 edition of the Google AI Challenge [5] was conducted with an RTS game named ANTS, in which the players control a set of ants that must 'fight' against the colonies of the rest of players in a grid with labyrinthine paths. The ants must gather food for generating new individuals and get an advance over the rivals. The fighting between ants is solved following some rules, but as a thumb rule, the higher number of ants are grouped, the easier will be to win a fight.

Thus, this is a RTS where the AI must be implemented at both commented levels: on the one hand, the ants must be grouped and specialized (explorers, fighters, gatherers), on the other hand each individual should have a particular behaviour to get a global emergent behaviour.

As a first approximation, a behavioural engine (for both levels) was designed by defining a set of states and rules guided by several parameters. This agent participated in the contest and finished in position 2076.

Then the initial engine has been improved by means of a Evolutionary Algorithms (EAs)[2]. They are a class of probabilistic search and optimisation algorithms inspired in darwinistic evolution theory. There are some types, including the extended Genetic Algorithms (GAs)[4], but the main features are common to all of them: a population of possible solutions (individuals) of the target problem, a selection method that favours better solutions and a set of evolutionary operators that act upon the selected solutions. After an initial population is created (usually randomly), the selection mechanism and the operators (crossover, mutation, etc) are successively applied to the individuals in order to create new populations that replace the older one. The candidates compete using their fitness (quality of adaptation). This process guarantees that the average quality of the individuals tends to increase with the number of generations. Eventually, depending on the type of problem and on the efficiency of the EA, the optimal solution may be found.

To conduct the evolution (in the evaluation step), every candidate agent in the population has fought against three different enemies (in two different approaches): a deterministic agent who finished in rank 993, and two very competitive agents which got position 1 and 165.

According to the results the agent has performed quite good, and has been able to beat bots which finished almost 1000 positions better than it in the competition.

2 State of the art

AI in games has become the most interesting element in actual games from the player's point of view, once the technical components (graphics and sound) have reached almost an upper bound. They mostly request opponents exhibiting intelligent behaviour, or just better human-like behaviours [9].

Researchers have also found it an interesting area from the early nineties, so this scope has presented an exponential grown in several videogames and fields, mainly starting with the improvement of FPS Bot's AI, the most prolific type of game [8, 11], and following with several games such as Super Mario [19], Pac-Man [10] or Car Racing Games [14], to cite a few.

The RTS games research area presents an emergent component [18] as a consequence of the commented two level AIs (units and global controllers). RTS games usually correspond to vast search spaces that traditional artificial intelligence techniques fail to play at a human level. As a mean to address it, authors in [15] proposed to extract behavioural knowledge from expert demonstrations which could be used to achieve specific goals. There are many research problems involving the AI for RTSs, including: planning with uncertainty or incomplete information, learning, opponent modelling, or spatial and temporal reasoning [1].

However, most of the RTS games in industry are basically controlled by a fixed script (i.e. a pre-established behaviour independent of inputs) that has been previously programmed, so they are predictable for the player some combats later. Falke et al. [3] tried to improve the user's gaming experience by means of a learning classifier system that can provide dynamically-changing strategies that respond to the user's strategies.

Evolutionary Algorithms (EAs), have been widely used in this field [16, 7], but they are not frequently used on-line (in real-time) due to the high computational cost they require. In fact, the most successful proposals for using EAs in games corresponds to off-line applications [17], that is, the EA works previously the game is executed (played), and the results or improvements can be used later during the real-time game. Through off-line 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. For instance, in [13] an agent trained with an EA to play in the previous Google AI Challenge is presented.

In the present work, EAs are also used, and an off-line Genetic Algorithm (GA) is applied to improve a parametrised behaviour model (set of rules), inside a RTS named ANTS.

3 The Google AI Challenge

This section describes the game scenario where the bots will play. The ANTS game was used as base for the Google AI Challenge 2011 (GAIC)[4] [6]. An ANTS match takes place on a map (see Figure 1) that contains several anthills. The game involves managing the ant community in order to attack (and destroy) the maximum number of enemy hills. Initially, game players have one or more hills and each hill releases the first ant. Then, the bot has to control it in order to reach food and generate another ant. Game is based on a turn system (1000 turns in official games). For each turn, participants have a limited time to develop a strategy with the ant community, i.e. decide the set of simple steps (just one cell in one direction) that every ants must perform. Before turn time-over, the bot should return a witness indicating that tasks have been finished. If the witness is not sent before time-over, the player receives the 'timeout' signal. This signal carries penalty points and the inability to make more movements until game finish. However, this does not entail game disqualification.

If the player has accumulated enough points before 'timeout', she could win. For each captured hill, the player receives two points and if one of our hills is captured, she misses a point.

[4] http://ants.aichallenge.org/

There are two strong constraints (set by the competition rules) which determine the possible methods to apply to design a bot: a simulated turn takes *just one second*, and the bot is *not allowed to store any kind of information between games* about its former actions, about the opponent's actions or about the state of the game (i.e., the game's map).

Thus, if desired, it is mandatory to perform an off-line (not during the match) fine-tuning or adaptation in order to improve an agent's behaviour. In this work, an evolutionary algorithm has been applied. Therefore, the goal in this paper is to design a bot/agent and improve it using an extra GA layer that consider a set of representative maps and enemies to train and adapt the bot for being more competitive, in order to fight the enemy, conquer its anthills, and finally win the game.

4 Algorithm and Experimental Setup

In this section the strategy to evolve is presented. A Genetic Algorithm (GA) is used to improve parameters of a basic agent. In order to improve the agent two different type of fitness functions and six different maps have been used.

4.1 Behavioural rules and parameters

The basic behaviour of our bot is mainly based in a Greedy strategy to prioritize multiple tasks entrusted to the ants:

```
IF enemy hill in sight
   attack the hill
ELSE IF food in sight
        pick up the food
  ELSE IF enemy ants in sight
             attack the ants
    ELSE IF non-explored zone in sight
                  explore the area randomly
```

The second part of the strategy, is a *lefty movement*, i.e. follow a straight line until water/obstacle is found, and then, walks to the left bordering it.

In order to perform a parameter optimization using genetic algorithms, we have defined a set in the above specified bot's rules. They are:

- *food_distance*: Maximum distance to go get food, i.e. ants ignores food that is at a distance greater than this value.
- *time_remaining*: Margin time we have for one turn to finish without a 'time-out penalty'. Higher values indicate that more actions are performed, but as previously explained, the player receives a penalty.
- *distance_my_ant_attack* and *distance_hill_attack*: These parameters are used to determine the attack priority. Distance_my_ant_attack means that we have one ant partner close enough to take advantage when attacking enemy ants.

In this situation, the distance_hill_attack is taking into account in order to change ant objective. If another enemy ant is close to our hill, our ant give priority to this more dangerous situation for our interest. In this case an ant is sacrificed to keep alive our anthill.
– *turns_lefty*: Maximum number of consecutive turns in which an ant lefty strategy can be used. After that number of turns, ants community change to Greedy strategy.

4.2 Genetic algorithm

A GA has been used to evolve the previously presented parameters. Thus each individual in the population is represented by an array of integers, where each number indicates the value of one of the parameters previously explained.

The *fitness function*, which determines the individual's adaptation to the environment, is based on launching a game against several opponents, in a certain number of turns and a specific map. The score for the agent after that game will determine the degree of kindness and individual adaptation to the problem we want to solve, knowing the individual that maximizes the score. Two different fitness functions have been studied:

– Basic fitness: it only considers the score obtained by our agent in the battle.
– Hierarchical fitness: the fitness is a tuple of the following elements in order: My score, enemy's score (negative), number of my own ants and number of enemy's ants (negative). A lexicographical order is applied to compare two individuals.

The considered operators have been:

– *selection*: choose half of population with individuals who obtained the highest scores in the games for improving the convergence component.
– *crossover*: multi-point crossover has been performed, mixing some parts of the parents to create the offspring.
– *mutation*: changes parameter values in an individual randomly (inside a range) with certain probability.

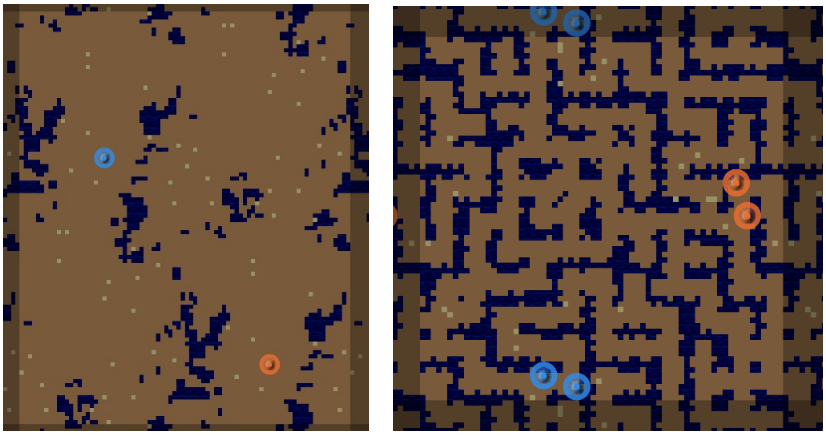
In order to achieve evolution it has been added an extra layer to the game implementation that allows us to store best individuals (set of parameters), and let to evolve the population in future generations.

4.3 Experimental setup

Six maps have been considered in order to perform the bot evolution. All of them are provided by the competition organizers in a tools package. Three maps are mazes with different level of difficulty and the rest are open walking areas. Figure 1 4.3 shows two examples of different type of maps. The circles mark hills positions with one colour for each team/player. The blue areas represent water that ants cannot cross, nor walk on it, small points represent food and the rest

Table 1. Maps

	Name	Type	#competitors	Rows	Cols	#Hills
map1	random_walk_p02_01	Open	2	100	80	1
map2	random_walk_p02_05	Open	2	52	70	1
map3	maze_p02_05	Maze	2	66	66	2
map4	maze_p02_34	Maze	2	108	138	1
map5	maze_p02_42	Maze	2	72	126	2
map6	cell_maze_p02_10	Open maze	2	42	142	2



(a) Map 1: open map (b) Map 3: maze/labyrinth type

Fig. 1. Two different example maps considered in the experiments.

are land where ants can move. Some other relevant information about maps is detailed in Table 1.

The experiments conducted try to analyze the performance of the implemented approaches (GA + fitness function) in each of the six maps. Both have considered 64 individuals in the population, a crossover rate equal to 0.3, a mutation rate of 0.1 and a stop criterion set to 20 generations. Every agent is evolved in the six maps 10 times in order to get a reliable fitness value; i.e trying to avoid the 'noisy nature' [12] of game playing as a valuation function for an individual when the opponent is non-deterministic. The reason is the same agent (individual) could be valued as very good or very bad depending on the combat result, which in turn depends on the enemy's actions and the game events.

5 Results and Analysis

Firstly it is important to notice that all the selected competitors which have been considered as opponents in the evolution got higher final rankings than our bot, who finished in rank 2076. They are a deterministic agent who finished in rank 993, and two very competitive and non-deterministic agents which got position

165 and the winner of the competition. Table 2 shows the obtained results in ten combats performed once the evolution has been completed.

Table 2. Results of ten battles between the evolved bot (using two fitness functions) and three different opponents with higher final ranks in the Google AI 2011 Competition. The scores, number of own and enemy's ants and the average number of turns to finish the match are presented, along with the standard deviation in each case.

	maxScore	**maxMyAnts**	**maxEnemyAnts**	**meanTurns**
	Basic fitness vs. Bot993.			
map1	3,00 ± 0,00	84,08 ± 43,82	66,50 ± 49,20	416,87 ± 125,94
map2	3,00 ± 0,00	68,08 ± 39,10	60,67 ± 34,92	425,64 ± 90,26
map3	6,00 ± 0,00	39,91 ± 15,65	186,91 ± 63,74	318,28 ± 124,63
map4	1,00 ± 0,00	8,64 ± 0,67	12,00 ± 0,00	150,00 ± 0,00
map5	5,42 ± 0,51	36,00 ± 27,24	228,75 ± 89,91	428,93 ± 189,53
map6	6,00 ± 0,00	46,25 ± 59,21	111,25 ± 20,82	221,18 ± 104,35
	Hierarchical fitness vs. Bot993.			
map1	3,00 ± 0,00	154,56 ± 28,84	2,67 ± 1,50	481,33 ± 48,26
map2	3,00 ± 0,00	97,67 ± 37,83	3,00 ± 2,18	486,78 ± 79,99
map3	6,00 ± 0,00	45,00 ± 8,85	118,33 ± 19,49	266,00 ± 55,57
map4	1,00 ± 0,00	9,22 ± 0,44	12,00 ± 0,00	150,00 ± 0,00
map5	4,67 ± 0,50	73,78 ± 71,10	226,89 ± 57,61	706,78 ± 262,88
map6	5,00 ± 1,15	104,11 ± 107,52	77,89 ± 58,10	519,44 ± 262,51
	Hierarchical fitness vs. Bot165.			
map1	0,00 ± 0,00	33,58 ± 2,97	101,17 ± 7,83	183,42 ± 7,29
map2	0,17 ± 0,39	31,08 ± 8,54	122,00 ± 49,41	221,17 ± 86,06
map3	0,00 ± 0,00	35,33 ± 9,72	98,83 ± 10,99	186,50 ± 9,26
map4	0,00 ± 0,00	34,75 ± 9,75	99,17 ± 10,96	184,92 ± 9,11
map5	0,00 ± 0,00	32,50 ± 10,51	101,75 ± 12,19	186,25 ± 9,18
map6	0,00 ± 0,00	31,50 ± 10,91	103,10 ± 12,80	188,00 ± 9,08
	Hierarchical fitness vs. Bot1.			
map1	0,00 ± 0,00	31,00 ± 34,00	109,00 ± 95,00	185,00 ± 198,00
map2	0,00 ± 0,00	17,00 ± 23,00	119,00 ± 132,00	156,00 ± 175,00
map3	0,00 ± 0,00	16,00 ± 17,00	118,00 ± 147,00	160,00 ± 186,00
map4	0,00 ± 0,00	14,00 ± 17,00	130,00 ± 120,00	166,00 ± 160,00
map5	0,00 ± 0,00	20,00 ± 31,00	112,00 ± 108,00	149,00 ± 147,00
map6	0,00 ± 0,00	21,00 ± 19,00	127,00 ± 131,00	172,00 ± 171,00

It could be noticed the small standard deviation present in most of the results, due to the small variations in the combat scores. It is zero in many cases because there are very few possible values (i.e. in maps with only two hills, max_score will be 0, 1 or 3 points). In addition, when a bot is good, it wins most of times and the other way round. Thus in the evolutionary process after 20 generations the system evolves always to reach max score.

For the same reason it can be seen in the table that our bot can not beat those in positions 165 and 1, since they are much more sophisticated in its defined behavioural engine. However, the evolution of the agent gets higher number of own ants and decreases the number of enemy ants.

Moreover, our evolved agent wins on all maps to the robot that ended in ranking 993, more than 1000 positions above the initial version (without optimization). The number of ants is the main difference between basic fitness and hierarchical fitness, and this feature allows to use more effective attack techniques. In maps 5 and 6, the score is lower than the obtained with basic fitness in some cases. However, the number of own ants doubles those obtained with a basic fitness. This invites us to improve strategies in such type of maps to achieve a better use of the large community of generated ants.

6 Conclusions and future work

This paper presents the design of an agent (bot) that plays in the RTS ANTS game proposed for the Google AI Challenge 2011. Starting with a combination of two basic behaviours (Lefty and Greedy) and a set of parameters, an Evolutionary Algorithm (EA) is used to fine-tune them and thus modify the agent's behaviour.

This bot is evolved in six maps provided by Google, and fighting three different bots that participated in the contest: those who finished in positions 993, 165 and the winner. Two different fitness functions have been tested: a basic function that only takes into account the final score (the number of conquered anthills in a run), and a hierarchical fitness, where the number of player's ants, turns, and enemy ants are also used to compare individuals.

Results show that, even evolving the parameters of two simple strategies, the agent is capable to win harder opponents. On the other hand, the same strategy is not affective against a medium-ranked bot, so it is clear that the enemy behaviour affects to the off-line training algorithms with an specific strategy. However genetic optimization is enough to beat a competitor who is above more than 1000 positions in the ranking.

We conclude that parameters optimization using EA significantly improves agent performance in RTS games and this technique would obtain better results combined with good planning strategies.

For future work, new combination of strategies will be studied and more different fitness funtions will be analysed: for example, combining all maps in each fitness calculation. Because the stochastic behaviour of some robots also affects the fitness, an study of how this fitness is affected during the algorithm run will be performed. As demonstrated, the behaviour of the enemies is also a very important key to analyse for designing a all-terrain bot: an agent should adapt to these different behaviours. Also, using a quick map analysis in each turn to set the parameters obtained in this work could be studied to adapt the agent accordingly. A map analysis could be performed, for example, counting the number of direction changes in a period of time. If many direction changes occurs by collisions with walls, means that bots are fighting in a map with maze pattern. Once map type has been detected, bot can choose suitable parameter group for the map. The combination of the Greedy and Lefty actions also will be studied in other RTS games, as the previous Google AI Contest games.

Acknowledgements

This work has been funded in part by projects P08-TIC-03903 (Andalusian Regional Government), and TIN2011-28627-C04-02 (Spanish Ministry of Science and Innovation), and by the FPU Grant 2009-2942.

References

1. M. Buro. Call for AI research in RTS games. In *Proc. AAAI workshop on Challenges in Game AI*, pages 139–141, 2004.
2. A. Eiben and J.E. Smith. What is an evolutionary algorithm? In G. Rozenberg, editor, *Introduction to Evolutionary Computing*, pages 15–35. Addison Wesley, 2005.
3. W. Falke-II and P. Ross. Dynamic strategies in a real-time strategy game. In *Proc Genetic and Evolutionary computational Conference (GECCO 2003)*, volume 2724 of *LNCS*, pages 1920–1921, 2003.
4. David E. Goldberg. *Genetic Algorithms in search, optimization and machine learning*. Addison Wesley, 1989.
5. Google. Google AI Challenge 2011: ANTS, 2011. Available at `http://aichallenge.org/`.
6. K.H. Holdum, C. Kaysø-Rørdam, and C. Østergaard. Google ai challenge 2011: Ants. *Jørgen Villadsen*, page 11, 2011.
7. 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 *Computational Intelligence and Games, 2009. CIG 2009. IEEE Symposium on*, pages 75–79. IEEE, 2009.
8. J. E. Laird. Using a computer game to develop advanced ai. *Computer*, 7(34):70–75, 2001.
9. L. Lidén. Artificial stupidity: The art of intentional mistakes. *AI Game Programming Wisdom 2*, pages 41–48, 2004.
10. E. Martn, M. Martnez, G. Recio, and Y. Saez. Pac-mAnt: Optimization based on ant colonies applied to developing an agent for ms. pac-man. In *2010 IEEE Conference on Computational Intelligence and Games (CIG 2010)*, pages 458–464, 2010.
11. A. M. Mora, M.A. Moreno, J.J. Merelo, P.A. Castillo, M.I. Garca-Arenas, and J.L.J. Laredo. Evolving the cooperative behaviour in Unreal^{TM} bots. In *Proc. 2010 IEEE Conference on Computational Intelligence and Games (CIG 2010)*, pages 241–248, 2010.
12. Antonio Miguel Mora, Antonio Fernández-Ares, Juan J. Merelo Guervós, Pablo García-Sánchez, and Carlos M. Fernandes. Effect of noisy fitness in real-time strategy games player behaviour optimisation using evolutionary algorithms. *J. Comput. Sci. Technol.*, 27(5):1007–1023, 2012.
13. Antonio Miguel Mora, Antonio Fernández-Ares, Juan Julián Merelo Guervós, and Pablo García-Sánchez. Dealing with noisy fitness in the design of a rts game bot. In *Applications of Evolutionary Computation - EvoStar 2012, Proceedings*, volume 7248 of *Lecture Notes in Computer Science*, pages 234–244. Springer, 2012.

14. E. Onieva, D.A. Pelta, J. Alonso, V., Milans, and J. Prez. A modular parametric architecture for the torcs racing engine. In *Proc. 2009 IEEE Symposium on Computational Intelligence and Games (CIG'09)*, pages 256–262, 2009.
15. S. Ontanon, K. Mishra, N. Sugandh, and A. Ram. Case-based planning and execution for real-time strategy games. In *Proceedings of the 7th international conference on Case-Based Reasoning: Case-Based Reasoning Research and Development (ISSBR'07)*, volume 4626 of *LNCS*, pages 164–178, 2007.
16. M. Ponsen, H. Munoz-Avila, P. Spronck, and D. W. Aha. Automatically generating game tactics through evolutionary learning. *AI Magazine*, 27(3):75–84, 2006.
17. P. Spronck, I. Sprinkhuizen-Kuyper, and E. Postma. Improving opponent intelligence through offline evolutionary learning. *International Journal of Intelligent Games & Simulation*, 2(1):20–27, February 2003.
18. P. Sweetser. Emergence in games. *Game development*, 2008.
19. J. Togelius, S. Karakovskiy, J. Koutnik, and J. Schmidhuber. Super mario evolution. In *IEEE Symposium on Computational Intelligence and Games (CIG'09)*, pages 156–161, 2009.