Co-Evolutionary Optimization of Autonomous Agents in a Real-Time Strategy Game

No Author Given

No Institute Given

Abstract. This paper presents an approach based in an evolutionary algorithm (EA), aimed to improve the behavioural parameters which guide the actions of an autonomous agent (bot) inside a real-time strategy game (RTS) named Planet Wars.
Specifically the work describes a co-evolutionary implementation of a previously presented method, which yielded successful results. Thus, it's analyzed the effects of considering several individuals to be evolved (improved) at the same time in the algorithm, the use of 3 different fitness for measure the goodness of each bot in the evaluation, and the variance of use an EA with and without previous knowledge for the training.
To this end, 4 on 4 matches have been considered. Two variants are presented: without previous knowledge (where the 4 bots belong to the population) and with (2 bots of the population versus 2 previously studied with good results bots).
For the fitness, 3 methods are studied: one based in turns and result position, and another two based in the survey of the percentage of ships that belong each bot in each turn of the battle.
In this paper we set several goals for the uses of the co-evolution: reduce the time needed for training in behavior with a huge time of evaluation, improve best bots for used in 4 on 4 battles fighting versus more bots, studies the significant different between the training with and without previous knowledge and finally studies many fitness for co-evaluations.

1 Introduction

Autonomous agents (or *bots*) in videogames have become very popular in the last years, because they can increase the challenge and lasting appeal of the game, by competing or cooperating with the human player. Thus, designing a good behavioural engine for them is one of the main topics of interest in the actual videogame development task. They have been widely used in Fisrt Person Shooter games (FPSs) from the nineties, but in this paper we will work with them on a Real-Time Strategy game (RTS). RTSs are a sub-genre of strategy-based video games in which the contenders control a set of units and structures distributed in a playing area and combat using them for conquering the scenario or defeating the opponent. Command and Conquer™, Starcraft™, Warcraft™ and Age of Empires™ are some examples of these type of games.

The RTS considered in the paper in named *Planet Wars*, and it was used a platform in the Google AI Challenge 2010. In this contest, 'real time' is sliced in one second *turns*, with players receiving the chance to play sequentially. However, *actions* happen at the *simulated* same time.

This paper describes a Co-Evolutionary [1] approach for improving the decision engine of a bot that plays that RTS. This engine consists in a set of rules previously designed and evolved by the authors in [2] , using an regular Genetic Algorithm (GA) [3]. We applied an offline evolution (i.e., not during the match, but prior to the game battles) of the parameters on which the behavioural rules depends.

The evaluation of the quality (fitness) of each set of rules in the population was made by playing the bot against predefined opponents, being a pseudo-stochastic or *noisy* function, since the results for the same individual evaluation may change from time to time, yielding good or bad values depending on the battle events and on the opponent's actions. We have dealt with this noisy nature [4] by means of a reevaluation phase of all the individuals every generation, along with an average calculation of the fitness value of every individual after five combats (in five different an representative maps).

The aim is that the co-evolutionary scheme improves the fitness convergence of the population, since the individuals cooperate in their evolution.

Thus, we have considered matches with four players, with two of the contenders the individuals being evolved at a specific generation, and the other two opponents with a fixed AI engine, namely the competition sparring *GoogleBot* in one of the experiments, and our best individual to date, baptised as *Genebot-8*, in the other one.

2 State of the Art

Video games have become one of the biggest sectors in the entertainment industry; after the previous phase of searching for the graphical quality perfection, the players now request opponents exhibiting intelligent behaviour, or just human-like behaviours [5].

Most of the researches have been done on relatively simple games such as Super Mario [6], Pac-Man [7] or Car Racing Games [8], being many bots competitions involving them.

RTS games show an emergent component [9] as a consequence of the cited two level AI, since the units behave in many (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 [10].

However, the reality in the industry is that in most of the RTS games, the bot is 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 becomes less interesting to play. In order to improve the users' gaming experience, some authors such as Falke et al. [11]

proposed a learning classifier system that can be used to endow 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 [12]. In this sense, Ontano et at. [13] proposed to extract behavioural knowledge from expert demonstrations in form of individual cases. This knowledge could be reused via a case-based behaviour generator that proposed advanced behaviours to achieve specific goals.

Evolutionary algorithms have been widely used in this field, 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 [14], 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 the present work, an offline GA is applied to a parametrised tactic (set of behaviour model rules) inside the Planet Wars game (a basic RTS), in order to build the decision engine of a bot for that game, which will be considered later in the online matches.

3 Problem Description

We work with a simplified version of the game Galcon, aimed at performing bot's fights which was used as base for the Google AI Challenge 2010 (GAIC)[1].

A Planet Wars match takes place on a map (see Figure 1) that contains several planets (neutral or owned), each one of them with a number assigned to it that represents the quantity of starships that the planet is currently hosting.

The aim of the game is to defeat all the starships in the opponent's planets. Although Planet Wars is a RTS game, this implementation has transformed it into a turn-based game, in which each player has a maximum number of turns to accomplish the objective. At the end of the match (after 200 actions, in Google's Challenge), the winner is the player owning more starships.

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* about its former actions, about the opponent's actions or about the state of the game (i.e., the game's map). Therefore, the goal in this paper is to design a function that, according to the state of the map in each simulated turn (input) returns a set of actions to perform in order to fight the enemy, conquer its resources, and, ultimately, win the game.

[1] http://ai-contest.com

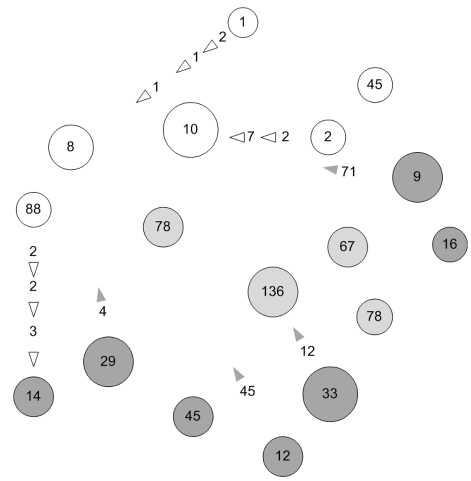

Fig. 1. Simulated screen shot of an early stage of a run in Planet Wars. White planets belong to the player (blue colour in the game), dark grey belong to the opponent (red in the game), and light grey planets belong to no player. The triangles are fleets, and the numbers (in planets and triangles) represent the starships. The planet size means growth rate of the amount of starships in it (the bigger, the higher).

In the original game, only two bots are faced but in this paper we studies what happen when we simulated 4 on 4 battles, it's mean, when 4 bots fight in the same map.

4 Co-Evolutionary Approach

We will resort to an generational co-evolutionary algorithm,

One of the major problems we found in the use of EAs for training bots for this behavior was the huge time needed for the evaluation, because we have to simulated all the battles, in several maps. In addition, we are forced to use re-evaluation between generations, when dealing with a noise and stochastic problem.

Theoretically, the use of a co-evolution allow reduce the number of simulations needed because it evaluates the population in "groups". That's, if we have a population of 100 individuals, in a classic GA we need make 100 evaluations, once for each individual, for each generation. In co-evolution case, for example, if we are using 2 bots of the population for the co-evolution we only need 50 evaluations. It's likely that 4-simulations will take more time that a 2-simulation, but the question is if the time taken for a "4-simulations" is less that the time taken for two "2-simulation". In that case, the co-evolution will decrease the time needed for the training.

The use of 2 individuals or 4 individuals of the population for the will depend of the use (or not) of previous knowledge.

4.1 Previous Knowledge vs Auto-generated Knowledge

We have a best training bot, which has proved its worth in 1vs1 battles. The question is: can we use our previous knowledge for improve (faster) better bots in a similar problem? Or maybe, it's best don't use previous knowledge and allow fight versus individuals of the population that theoretically (as the bases of the GAs) will be better and better in each generation? For answer this question, we will make two types of co-evolutions.

Co-evolution with previous Knowledge In this case, we will simulated battles between two individuals of the population versus two of our best bots (in 1vs1). We expected that our co-bots can learn the bases of our best bots, and improve for be better in 4 on 4 battles. It's desirable that this type of co-evolution prize bots that at least can win in a battled of our best bots.

Talcking about the time needed for the execution, we don't expected an huge reduction of the time needed for the training of the bots. It's something that we will try to get in the next co-evolution method.

Co-evolution with Auto-generated Knowledge In this case, we will simulated battles between four individuals of the population. We don't use previous knowledge of our previous works, but we are improving our bots for be better versus bots every time betters (because we expected that our population will better in each generation).

Talcking about the time, we expected that this method reduce the time needed near to the half that the previous method.

4.2 Fitness

In our previous works, we evaluated a single bot of the population versus always the same bot (an reference-bot). For fitness function, the bot was evaluated several times (in different maps). The fitness function was defined depending of the result of the battle (if the bot win all his battles or lose in someone) and the numbers of turns needed for end the game. For two bots of the population (A and B) the fitness is defined like show in Fig.2

This fitness works well for battles between two bots, and our first fitness proposed is an natural evolution of this fitness for 4 bots battles.

Fitness based in Position-turns This fitness it's an natural evolution of the previous fitness, applied to 4 bots battles. Again, the evaluations are in several maps. In this case, we studie the position (1th,2th,..) of the bot in the 4-battle and the number of turns. For a bot that wins all the battles (it's 1th in all the battles) we call the sum of the numbers of turns $ferocity$, and in previous works we found that a bot that wins in less turns it's best that other that takes more turns (even if the both wins). In other case, the sum of turns is call *sturdy* and oposite to the $ferocity$, it's desirable a bot that take more turns in be defeated.

```
A, B ∈ Population
if A WINs always then
    if B LOSEs some battle then
        A is better than B
    else if A take less turns than B then
        A is better than B
    else
        B is better than A
    end if
else
    if B WINs always then
        B is better than A
    else if A take less turns than B then
        B is better than A
    else
        A is better than B
    end if
end if
```

Fig. 2. Fitness used in battles of 2 bots

```
A, B ∈ Population
if A average position < B average position then
    A is better than B
else if A average position > B average position then
    B is better than A
else
    if A,B is always 1th then
        if A take less turns than B then
            A is better than B
        else
            B is better than A
        end if
    else
        if A take less turns than B then
            B is better than A
        else
            A is better than B
        end if
    end if
end if
```

Fig. 3. Fitness used in battles of 4 bots based in turns and results

The principal problem in the previous fitness, it's that there we are using two independents variables. In this case, we try to found a fitness that resume the goodness of our bot in a single number.

In this fitness, we are only interesting in the final result: position and turn. We aren't student how the bot reach it. In the nexts fitness, we use another metric to define the goodness of the bots. The percentage of the ships that in each turn belong to each player.

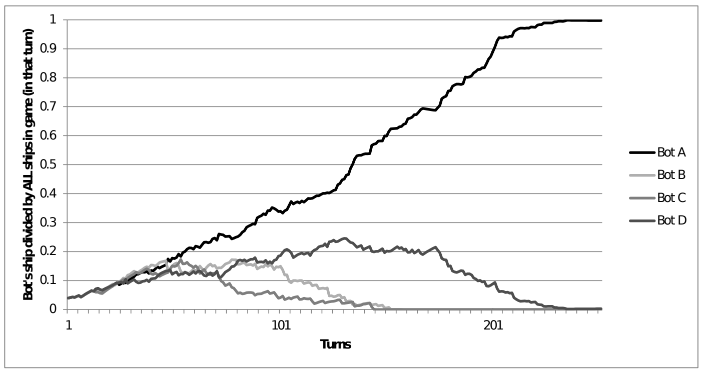

Fig. 4. Representation of the number if ships of each bot in each turn

Fitness based in Slope For this fitness, we use the leats squares regression analysis for resume the cloud of points to a simple rect. The rect is represented as $y = \alpha \times x + \beta$ where, α and β are calculated as show the Fig. 5:

$$\alpha = \frac{\sum_{i=1}^{n}(X_i - \bar{X_i})(Y_i - \bar{Y_i})}{\sum_{i=1}^{n}(X_i - \bar{X_i})^2} \tag{1}$$

$$\beta = \bar{Y} - \alpha\bar{X} \tag{2}$$

Fig. 5. Leats squares regression

For our fitness, we take the slope of the rect: α. We can see if the bot win, $\alpha > 0$ and if loses, $\alpha < 0$. However,

How we are using several evaluations in different maps,

4.3 Fitness bases in Integral-Area

For the co-evolution of the bots

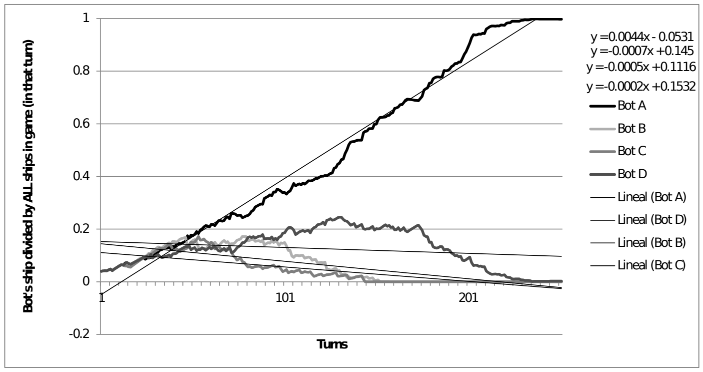

Fig. 6. Representation of the number if ships of each bot in each turn

5 Experiments and Results

6 Conclusions and Future Work

Acknowledgements

This paper has been funded in part by projects P08-TIC-03903 (Andalusian Regional Government), TIN2011-28627-C04-02 (Spanish Ministry of Science and Innovation), and project 83 (CANUBE) awarded by the CEI-BioTIC UGR.

References

1. Paredis, J.: Coevolutionary computation. Artificial Life **2** (1995) 355–375
2. Fernández-Ares, A., Mora, A.M., Merelo, J.J., García-Sánchez, P., Fernandes, C.: Optimizing player behavior in a real-time strategy game using evolutionary algorithms. In: Evolutionary Computation, 2011. CEC '11. IEEE Congress on. (2011) 2017–2024
3. Goldberg, D.E.: Genetic Algorithms in search, optimization and machine learning. Addison Wesley (1989)
4. Mora, A.M., Fernández-Ares, A., Merelo, J.J., García-Sánchez, P.: Dealing with noisy fitness in the design of a RTS game bot. In: Proc. Applications of Evolutionary Computing: EvoApplications 2012, Springer, LNCS, vol. 7248 (2012) 234–244
5. Lidén, L.: Artificial stupidity: The art of intentional mistakes. In: AI Game Programming Wisdom 2, Charles River Media, INC. (2004) 41–48
6. Togelius, J., Karakovskiy, S., Koutnik, J., Schmidhuber, J.: Super mario evolution. In: Proceedings of the 5th IEEE Symposium on Computational Intelligence and Games (CIG'09), Piscataway, NJ, USA, IEEE Press (2009)
7. Martín, E., Martínez, M., Recio, G., Saez, Y.: Pac-mant: Optimization based on ant colonies applied to developing an agent for ms. pac-man. In Yannakakis, G.N., Togelius, J., eds.: Computational Intelligence and Games, 2010. CIG 2010. IEEE Symposium On. (2010) 458–464

8. Onieva, E., Pelta, D.A., Alonso, J., Milanés, V., Pérez, J.: 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) 256–262
9. Sweetser, P.: Emergence in Games. Game development. Charles River Media, Boston, Massachussetts (2008)
10. 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)
11. 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)
12. Aha, D.W., Molineaux, M., Ponsen, M.: Learning to win: Case-based plan selection in a real-time strategy game. In: in Proceedings of the Sixth International Conference on Case-Based Reasoning, Springer (2005) 5–20
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. Spronck, P., Sprinkhuizen-Kuyper, I., Postma, E.: Improving opponent intelligence through offline evolutionary learning. International Journal of Intelligent Games & Simulation **2**(1) (2003) 20–27