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

A. Fernández-Ares, A.M. Mora, M.G. Arenas, J.J. Merelo,
P. García-Sánchez, and P.A. Castillo

Departamento de Arquitectura y Tecnología de Computadores.
Universidad de Granada (Spain)
antares.es@gmail.com,
{amorag,jmerelo,pgarcia,pedro}@geneura.ugr.es

Abstract. This paper presents an approach based in an evolutionary algorithm, aimed to improve the behavioral parameters which guide the actions of an autonomous agent (bot) inside the real-time strategy game, Planet Wars. The work describes a co-evolutionary implementation of a previously presented method GENEBOT, which yielded successful results, but focused in 4vs matches this time. Thus, there have been analyzed the effects of considering several individuals to be evolved (improved) at the same time in the algorithm, along with the use of three different fitness functions measuring the goodness of each bot in the evaluation. They are based in turns and position, and also in mathematical computations of linear regression and area regarding the number of ships belonging to the bot/individual to be evaluated. In addition, the variance of using an evolutionary algorithm with and without previous knowledge in the co-evolution phase is also studied, i.e., respectively using specific rivals to perform the evaluation, or just considering to this end individuals in the population being evolved. The aim of these co-evolutionary approaches are mainly two: first, reduce the computational time; and second find a robust fitness function to be used in the generation of evolutionary bots optimized for 4vs battles.

1 Introduction

Planet Wars is a very famous Real-Time Strategy game (RTS), introduced in Google AI Challenge 2010 [1] as a framework where the contenders could create their own autonomous players (bots). This is a pseudo-turn-based game, rather than a real-time one, since it considers one second micro-turns for the bots decide their next set of actions, which then happen at the same simulated time. The competition consisted in develop the best bot regarding the number of matches it won against the rivals.

Our first approach in this field was the so-called GENEBOT [1], which is an evolutionary approach for the improvement of a bot's decision engine. It consists in a set of parametrised rules that model the behaviour of the bot, and which

[1] http://planetwars.aichallenge.org/

was defined by an expert human player. Then, a Genetic Algorithm (GA) [2] was applied offline (not during the game) for evolving these parameters.

GENEBOT was designed and optimized for 1 vs 1 battles, using a turn-based fitness function. It evaluated all the individuals in the population by playing five different matches (in five different maps) against a sparring bot provided by the competition (GoogleBot), trying to avoid with these repetitions the noisy factor present in this problems (videogames) [3], since the fitness value could be very good or very bad for the same individual in different matches. The reason is that it depends on the pseudo-stochastic opponent's actions, and also on its own non-deterministic decisions.

In this paper, the aim is to face 4vs matches, trying to define an evolutionary approach which improves the cited behavioural engine (set of rules) to this type of play. Thus, a co-evolutionary method has been proposed.

Co-evolutionary algorithms (CEA) [4] are those which consider different groups of potential solutions (individuals) evolving inside an environment, sometimes interacting with it, and at the same time, being affected by it. Every set of individuals which interact with the environment in the same way is called a *specie*. In CEAs several species (usually two) are able to live and evolve in the same environment grouped in populations.

The fitness of the individuals is usually calculated using some individuals of other populations, according to the dependencies between species (interactions among populations). Depending on these interactions the CEAs are classified in:

- *Competitive co-evolutionary algorithms* [5], where the fitness of an individual depends on competition with other individuals from other species.
- *Cooperative co-evolutionary algorithms* [6], where the aim is to find individuals from which better environment can be constructed. The fitness of an individual depends on its ability to cooperate in solving the target problem with individuals from other species.

In this work, both types have been implemented since the evolution is performed following two different shapes for the four contenders in every match (in the evaluation step): on one approach two of the contenders are individuals being evolved at a specific generation, and the other two opponents with a fixed Artificial Intelligence (AI) engine. On the other approach, four different players corresponding to four individuals being evolved are considered. These are respectively baptised as *previously knowledge-based* approach and *non-previously-knowledge-based* one. So, according to the previous taxonomy, both algorithms are competitive in the fitness calculation step and cooperative regarding the problem solving (find the best agents for this type of battles).The aim of using co-evolution is mainly to reduce the computational time, since a set of individuals are evaluated at a time (two or four, depending on the approach).

Moreover, three different fitness functions are also tested in the work, having three different types: one based in the bot's position and number of turns; another one based in the computation of a linear regression based on the percentage of ships with respect to the total; and the last calculating the integral of the function which represents these numbers. All of them consider the final results of every individual (bot) after the aforementioned five matches (on average).

Several experiments have been conducted analyzing the robustness of the different fitness functions, and the influence and performance of the previous knowledge consideration (or not) in the evolution.

2 State of the Art

Evolutionary Computation (EC) has been widely applied in the videogames area, including the RTS games, in different issues, such as: automated tactics generation [7], decision making improvement [8], or parameter optimisation in behavioural engines [9, 10]. This work is enclosed in the latter category, since it considers a set of rules, initially defined by an expert, which model a bot's AI in the RTS Planet Wars. These rules are optimised regarding the set of parameters which determine how the bot behaves. This kind of improvement was previously performed by the authors in [1, 11, 12] by means of off-line (before the game) Evolutionary Algorithms. In the present paper the optimisation have been performed using a Co-Evolutionary approach.

Co-Evolutionary Algorithms (CEAs) have been previously used in this scope, initially regarding puzzle and board games such as Backgammon [13], or Go [14]. The first work proposed a very simple hillclimbing algorithm to evolve a population of neural networks, playing among them as rivals, in a competitive co-evolutionary approach. The latter paper presented a co-evolutionary learning approach which performed well once the EA was correctly tuned, moreover, this method yields better players to solve small Go boards since every individual is evaluated against a diverse population of rivals. In the same line, there are some other works in the card games area, such as [15], aimed to create Poker agents, considering a co-evolution process in which the players are part of the learning process. This meant a difficult process to get robust strategies, due to the variation in opponents, but the results shown to fit with some recommended strategies according experts. The aim of the present work is to conduct a study implementing a similar co-evolutionary approach, being competitive in the fitness calculation, but cooperative since all the opponents are also part of the same learning process (same population).

In recent years, this type of EAs has been also applied to videogames, enclosed in the Computational Intelligence (CI) branch of AI. For instance Togelius et al. [16] studied the co-evolution effects of some populations in car racing controllers, comparing the performance of a single population against various, implementing both generational and a steady-state approaches. Avery and Michalewicz introduced in [17] a co-evolutionary algorithm (for the game TEMPO) which used humans as rivals for the individuals in the evolutionary process. Cook et al. [18] presented a cooperative co-evolutionary approach for the automated design of levels in simple platform games. And recently Cardona et al. [19] studied the performance of a competitive algorithm for the simultaneous evolution of controllers to both Ms. PacMan and the Ghost Team which has to chase her.

Regarding the RTS scope, there are also several related works, such as the study by Livingstone [20], who compared several AI-modelling approaches for

RTS games, and proposed a framework to create new models by means of co-evolutionary methods. He considered two levels of learning in a hierarchical AI model (inside an own-created RTS), evolving at the same time different partners in different strategic levels, so it was a cooperative approach. It is different to the one proposed here, since in the present work the co-evolution occurs at the same level for all the individuals. The work by Smith et al. [21] presents an analysis on how a co-evolutionary algorithm can be used for improving students' playing tactics in RTS games. Other authors proposed using co-evolution for evolving team tactics [22]. However, the problem is how tactics are constrained and parametrised and how the overall score is computed. Nogueira et al. [23] considered in a recent publication the use of a Hall of Fame as a set of rivals (in the evaluation function) inside a co-evolutionary algorithm to create autonomous agents for the RTS RobotWars.

The latter is an approach based in a self-learning algorithm as the one we are proposing, but focused in a subset of individuals (the elite) which can have a negative effect in the generalisation factor or the bots' knowledge. We have tested our method considering different rivals, in different studies. One of them is a previously optimized (and good-performing) bot, in order to deal with the so-called previous knowledge (Section 4.1).

3 Problem Description

In this paper we works with a version of the game Galcon, aimed at performing bot's fights which was used as base for the Google AI Challenge 2010 (GAIC)[2].

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

The aim of the game is to defeat all the ships 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, the winner is the player that remains alive, or that which owns more ships if more than one survives.

There are two strong constraints 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 aim in this paper is to study the improvement of a bot according to the state of the map in each simulated turn (input), returning a set of actions to perform in order to fight the enemy, conquering its resources, and, ultimately, wining the game. In the original game, only two bots are faced but in this paper it is studied what happen when we simulate 4 on 4 battles, i.e., when 4 bots are fighting in the same map.

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

4 Cooperative and Competitive Evolution: Co-Bots

There are two types of co-evolution attending to the interaction between the individuals in the population: cooperative and competitive. In this paper, it is described a co-evolutionary algorithm that considers both sides.

Thus, on the one hand, there are simulated 4 on 4 battles where several bots fight for the same goal: win the battle. That means that every bot competes against each other, or in EA-based concepts, every individual in the population competes for perpetuating its species. On the other hand, the cooperative factor arises because every 4 on 4 battle is *all versus all*, so a single individual could have assistance from others for *killing* a (temporary) common rival, since at the end, only one must remain (the winner).

In addition, in the GA the individuals are sharing *knowledge*, because they are *living* (and fighting) together. If the population is improved, the new offspring *born* of the previous parents will behave better.

One of the main problems of using EAs for training bots in this problem is the huge amount of computational time needed for the evaluation, since it consists in a simulation of several complete battles (in five different and representative maps), moreover a reevaluation process will be performed in every generation (of every individual). The aim of both processes is dealing with the noisy nature of the fitness function due to the pseudo-stochasticity of the problem [3].

Theoretically, the use of a co-evolutionary approach would allow to reduce the number of simulations needed, because the individuals are evaluated in *groups*.

For instance, in a population of 100, the use of co-evolution considering two individuals per evaluation would reduce the number of evaluations in a half.

In our case, we will test two different approaches, considering both *4 bots simulations* and *2 bots simulations*. The use of two or four individuals of the population in the experiments depends respectively on the use (or not) of previous knowledge.

4.1 Previous Knowledge vs Auto-generated Knowledge

In this work we understand as previous knowledge the consideration of a rival evolved in a previous study [24], *GENEBOT*, which proved to be a very competent rival in 1 on 1 battles. This bot will be used as a part of the evaluation process in 4vs battles. The aim is to study the influence of this so-called 'previous knowledge' in the performance of the proposed Co-Evolutionary Genetic Algorithm (Co-GA). To this end, several experiments have been conducted considering an approach in which two of the rivals are GENEBOTs, and some others in which all the rivals are individuals of the population being evolved.

Thus, two main approaches have been considered:

- **Co-evolution with Previous Knowledge**: In this case, battles between two individuals of the population versus two of the best bots (GENEBOTs) have been simulated in the evaluation process. It is expected that the Co-Bots are able to learn the basis of the GENEBOTs, and improve for be better

rivals in 4 on 4 battles. To this end the algorithm rewards bots that, at least, win in a battle against GENEBOTs. Since the approach will evaluate Co-Bots in groups of two, the running time of this approach, will not be reduced in a great factor during the training phase of the bots.

– **Co-evolution with Auto-generated Knowledge**: In this case, battles with four individuals of the population have been tested. For this approach the knowledge is included into the individual when it fights in previous battles, i.e., the rivals are included in the learning process. In this approach a considerable running time reduction in the comparison with the previous proposal is expected, because the number of evaluations has been reduced to the half.

4.2 Fitness Functions

In previous works, a bot was evaluated always versus the same bot (a reference bot), several times (in different maps). The fitness function is defined depending of the result of the battle (if the bot wins all its battles or loses in any of them) and the number of turns needed for ending the game. For two bots, A and B the fitness was defined as Fig. 1a shows.

This fitness works well for 1vs1, but in this work we have redefined it for 4vs matches. Moreover two additional evaluation functions have been proposed.

Fitness based in Position and turns. This fitness is the natural evolution of the previous one, applied to 4 bots battles. Again, the evaluations are done in several maps. In this case, both the position (1^{th} to 4^{th}) of the bot in the battle, and the number of turns needed, are included into the formula.

We define the term $ferocity$, regarding a bot that wins all its battles. This factor is included in the fitness computation as the sum of turns the bot has needed to win. This sum is considered to select the best bot when more than one wins all its battles, since a bot that wins in less turns is better than other that wins needing more. In some other cases, the sum of turns is called *sturdy*, and opposite to the $ferocity$, it is desirable a bot that take more turns in being defeated. In Fig. 1b there is a formal definition of this fitness.

In this fitness, we are only interested in the final result (position and number of turns). We do not include in the analysis how the bot has reached them. The problem of this function is that the consideration of two different terms makes it difficult the comparison between different evaluations. In order to let easier comparisons two other fitness functions have been defined.

Both of them are based in the percentage of ships belonging to each player in every turn. They are normalized considering the total number of ships in the game for that turn (including neutrals ships in neutral planets). For each player, we have a different *cloud* of ships.

Below, are described the two alternatives to deal with this cloud of points for the fitness function: the use of slopes and areas.

```
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
```

(a) Fitness considered in 1 vs 1 battles

```
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 1^th 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
```

(b) Fitness considered in 4vs battles

Fig. 1: Fitness functions based in turns and positions

Fitness based in Slope. For this fitness, a square regression analysis is computed in order to transform the cloud of points into a simple line. The line is represented as $y = \alpha \times x + \beta$, where α and β are calculated as shown in Equations 1 and 2, computing a least squares regression. For every bot in the simulation we calculate α and (*slope*). This *slope* is the fitness of every bot for that simulation. A graphical example can be seen in Fig. 2.

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

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

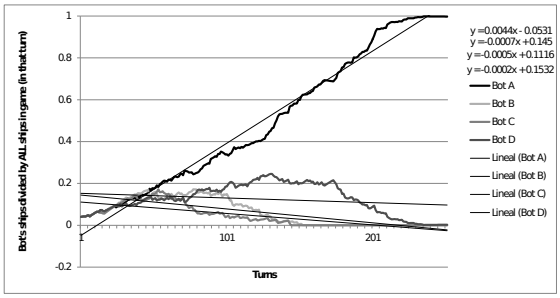


Fig. 2: Fitness based in Slope: number of ships of every bot in each turn

Theoretical maximum and minimum values are set for this fitness. An optimum bot that wins in the first turn, has an ideal slope of 1, so this is the maximum value of our fitness. On the other hand, a bot that loses in the first turn, has a slope of -1. Thus, if we calculate the *slope*, we know if the bot $WINs$ ($slope > 0$) or $LOSEs$ $slope < 0$. The values of the different battles are summed to compute the global *slope*. Then, the bot with the highest value will be the best is each turn or battle.

4.3 Fitness based in Area.

In this case, the integral of the curve of the bot's live-line is used for calculating the area that is 'covered' by the fitness cloud of points (see Equation 3). This *area* is normalized considering the number of turns, and thus it represents the average percentage of ships during the battle for each player.

$$area = \frac{\int_0^t \%ships(x)dx}{t} \tag{3}$$

As in previous case, maximum and minimum values has been set for this fitness. If an optimal bot wins in the first turn, the area of each live-line is close to 1, so this is the maximum value of the fitness. Otherwise, if a bot loses in the first turn, its live-line area is close to 0. In this case, we do not extract additional information related with which bot wins the battle, because the area of the live-line is not related with the winner of the battle. Thus, we are losing some information.

5 Experiments and Results

Several experiments have been conducted in order to study different issues of the proposed approaches, but having in mind that the main objective is the improvement of bots using a co-evolutionary algorithm. The set of parameters considered in the Co-GA is shown in Table 1. For each one of the presented approaches (combinations of fitness functions and knowledge-related methods), ten executions of the Co-GA have been performed.

Population Size	Generations	Crossover prob.	Mutation prob.	Elitism	Tournament size
100	200	0.6	0.1	20	2

Table 1: Table with the parameters used in the Co-GA.

As a first set of results, the average time consumed in the run of every generation is plotted in Fig. 3. The values are, as expected, reduced in a half in the previous knowledge approach (2 individuals per evaluation), and in 3/4 in the auto-generated knowledge (4 individuals per evaluation). This could be non-surprising results, but there should be considered than the original method evaluated 1vs1 battles, which are usually finished in less time than 4vs matches.

The second analysis concerns the fitness evolution. To this end the best, worst, median and average (with standard deviation) fitness values are studied. In Fig. 4 it can be seen the corresponding values to the best execution in every case, i.e. that of the 10 runs which obtained the considered as the best bot.

In auto-generated knowledge fitness (Fig.4b) it can be observed an homogeneous distribution throughout the generations, because the individuals are learning all of all, so the differences among them remain equal (in some limits) along the whole evolutionary process. That figure shows a high variability (or oscillations) in the best fitness graph, due to the rivals (in evaluation) are included in

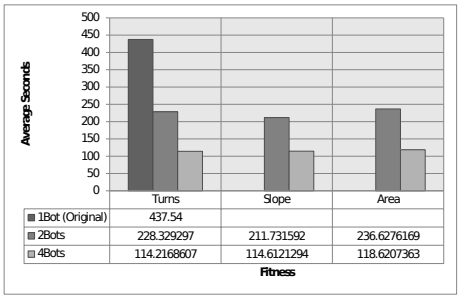


Fig. 3: Average time per generation (secs)

the learning process, so the population is not adapted to a common behavioural pattern, but every individual must learn to compete against (potentially) N-1 different bots. This means that the best individual varies frequently.

In the case of previously knowledge fitness (Fig.4a), it can be noticed a slight improvement, on the average fitness in the first generations. This happens because more and more individuals are able to beat the bots considered as rivals (GENEBOTs), an expected result. In addition, and due to this behavioural pattern to fight with, the evolution of the best fitness presents less variations along the generations.

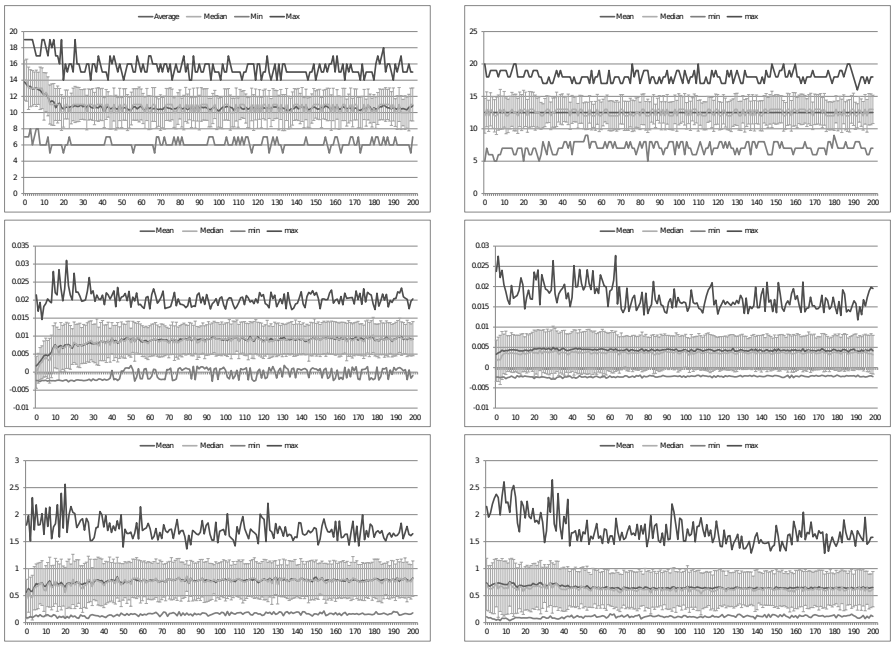
Anyway, the noisy nature of the fitness function due to the problem itself [3], remains here and this means that traditional evolution graphs (clear fitness improvement with very slight variations along generations) could not be obtained.

Next a comparison between the methods has been performed. However, since the fitness computation function is different for each method, they can not be directly compared (as they have different magnitudes). So it is necessary to perform simulations in a wide range of maps and enemies to study how good they are.

First a fool bot (GoogleBot) has been chosen to simulate 400 battles. In these battles, every bot obtained by Co-GA has to fight against 3 of these bots. The results have been excellent, with 100% of victories of the Co-Bots against them.

Then, a tough set of rivals has been considered, being three copies of the best bot previously obtained in other work, namely three GENEBOTs. 500 battles against them have been simulated, and the results are shown in Fig.5. The best average results are obtained for the Co-Bots obtained using previous knowledge, since they were trained to deal with at least two of them during the evolution. However, the scenario here is quite different since there is another GENEBOT as rival. As a general result, in both cases, the worst performance is achieved considering turns/position based fitness.

To study whether the bots actually achieved by the turn-based fitness are really less competitive, we have conducted a simulation (3 vs 3) matching the three bots achieved by the three fitness methods using, separately per knowledge approach. Fig.6a shows the results for each knowledge method. It is complemented



(a) Methods with previous knowledge: from top to down Turns/Position, Slope and Area

(b) Methods with auto-generated knowledge: from top to down Turns/Position, Slope and Area

Fig. 4: Best, worst, average, median and Std.Dev for fitness in every generation.

with the summed results plotted in Fig.6b, which shows the sum per fitness method (for the two knowledge approaches). As it can be seen the bots obtained with the fitness based in turns/positions perform clearly worse than the others.

Finally, the top four of the yielded bots are tested in 500 battles between them. There is one bot per fitness function, without considering the turn-based ones (since they are worse). Fig.7 shows how the bots have performed and their position in the battles. As it can be seen Slope-based Co-Bots are better than Area-based ones. Similarly, bots trained with previous knowledge, perform better than those with auto-generated knowledge.

6 Conclusions and Future Work

This paper presents some cooperative co-evolutionary approaches for improving autonomous agents for playing the RTS game Planet Wars. They are trained for 4 on 4 battles, considering two types of evaluation: based in previous knowledge (i.e. against a previously obtained bot), and with auto-generated knowledge (i.e. the rivals are included in the learning process). In addition three different fitness functions have been tested for each method: a position/turns based one, linear

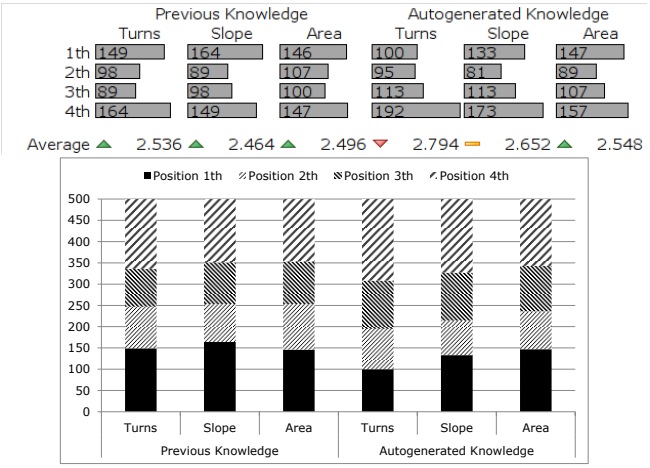
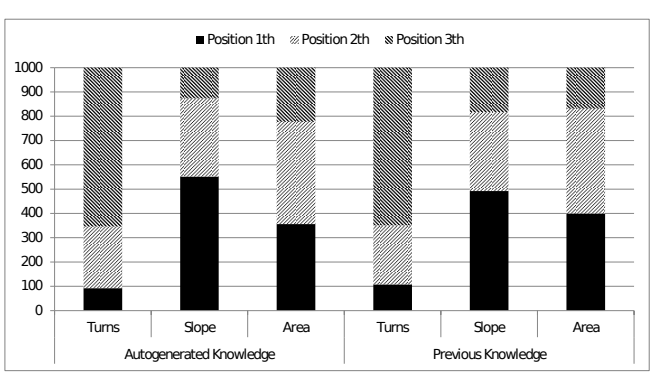
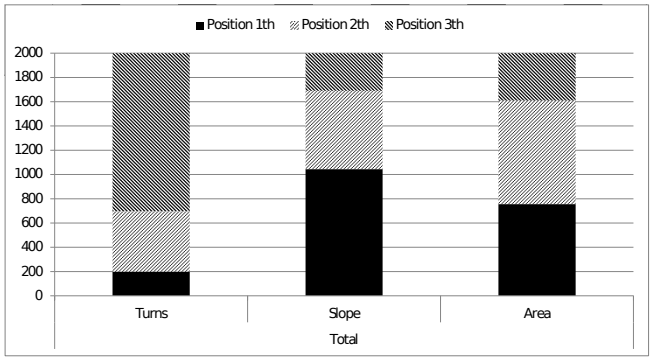


Fig. 5: Results (positions) of 500 4vs battles between 3 GENEBOTs and a bot per each method.



(a) Results (positions) per knowledge approach



(b) Total sum of results (same positions) per fitness method, considering both knowledge approches

Fig. 6: Results of 1000 3 vs 3 battles between the obtained bots.

regression model and area computation, regarding the number of ships belonging to the bot/individual to be evaluated in the last cases.

The first remarkable result is the significant reduction of the training time needed, due to the use of co-evolution.

It has been shown that a position/turn-based fitness function is less effective for training than one based on the study of curve resources (ships in this paper), in co-evolutionary approaches. Regarding the two mathematical fitness, the one based on the slope has proven to be slightly better, possibly due to a better representation of the bots victories against losses. Another interesting result point to that the use of previous knowledge can make a difference, but not very

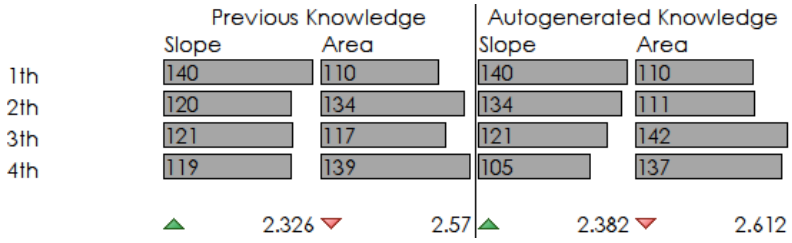


Fig. 7: Results (positions) of 500 4vs battles matching the four best bots obtained (one per approach).

significant. In the future it may be interesting to go deep in the study on the real influence of the previous knowledge.

Moreover this paper opens up new lines of research on the proposed problem. Such as the study of different ways for extracting knowledge from the population for co-evolution, for example the use of a Hall of Fame as proposed by the authors in [23], or the study of other mathematical fitness approximations. Regarding the co-evolution, it could be studied a competitive approach in which there could be different subpopulations devoted to improve different controllers for agents.

Acknowledgements

The authors would like to thank the FEDER of European Union for financial support via project "Sistema de Información y Predicción de bajo coste y autónomo para conocer el Estado de las Carreteras en tiempo real mediante dispositivos distribuidos" (SIPEsCa) of the "Programa Operativo FEDER de Andalucía 2007-2013" . We also thank all Agency of Public Works of Andalusia Regional Government staff and researchers for their dedication and professionalism.

References

1. 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
2. Goldberg, D.E.: Genetic Algorithms in search, optimization and machine learning. Addison Wesley (1989)
3. Anonymous: Our title - noisy fitness. One Journal **vol**(num) (20XX) pp
4. Paredis, J.: Coevolutionary computation. Artif. Life **2**(4) (1995) 355–375
5. Rosin, C.D., Belew, R.K.: New methods for competitive coevolution. Evol. Comput. **5**(1) (1997) 1–29

6. Potter, M., Jong, K.: A cooperative coevolutionary approach to function optimization. In Davidor, Y., Schwefel, H.P., Männer, R., eds.: Parallel Problem Solving from Nature - PPSN III. Volume 866 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (1994) 249–257
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. Mora, A.M., Ángel Moreno, M., Merelo, J.J., Castillo, P.A., Arenas, M.I.G., Laredo, J.L.J.: Evolving the cooperative behaviour in Unreal™ bots. In Yannakakis, G.N., Togelius, J., eds.: Computational Intelligence and Games, 2010. CIG 2010. IEEE Symposium On. (2010) 241–248
10. Fernández-Ares, A., Mora, A.M., Merelo, J.J., García-Sánchez, P., Fernandes, C.M.: Optimizing strategy parameters in a game bot. In: Proc. 11th International Work-Conference on Artificial Neural Networks, IWANN 2011, Springer, LNCS, vol. 6692 (2011) 325–332
11. 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
12. Fernández-Ares, A., García-Sánchez, P., Mora, A.M., Guervós, J.J.M.: Adaptive bots for real-time strategy games via map characterization. In: 2012 IEEE Conference on Computational Intelligence and Games, CIG 2012, IEEE (2012) 417–721
13. Pollack, J.B., Blair, A.D.: Co-evolution in the successful learning of backgammon strategy. Machine Learning **32** (1998) 225–240
14. Runarsson, T.P., Lucas, S.M.: Co-evolution versus self-play temporal difference learning for acquiring position evaluation in smallboard go. IEEE Transactions on Evolutionary Computation **9**(6) (2005) 628–640
15. Thompson, T., Levine, J., Wotherspoon, R.: Evolution of counter-strategies: Application of co-evolution to texas hold'em poker. In: IEEE Symposium on Computational Intelligence and Games (CIG'08), IEEE (2008) 16–22
16. Togelius, J., Burrow, P., Lucas, S.M.: Multi-population competitive co-evolution of car racing controllers. In: Proceedings of the IEEE Congress on Evolutionary Computation (CEC). (2007) 4043–4050
17. Avery, P.M., Michalewicz, Z.: Adapting to human game play. In: IEEE Symposium on Computational Intelligence and Games (CIG'08). (2008) 8–15
18. Cook, M., Colton, S., Gow, J.: Initial results from co-operative co-evolution for automated platformer design. In: Applications of Evolutionary Computation, EVOApplications 2012, LNCS 7248. (2012) 194–203
19. Cardona, A., Togelius, J., Nelson, M.: Competitive coevolution in Ms. Pac-Man. In: Proceedings of the 2013 IEEE Congress on Evolutionary Computation (CEC 2013). (2013) 1403–1410
20. Livingstone, D.: Coevolution in hierarchical ai for strategy games. In: IEEE Symposium on Computational Intelligence and Games (CIG'05), IEEE (2005)
21. Smith, G., Avery, P., Houmanfar, R., Louis, S.: Using co-evolved rts opponents to teach spatial tactics. In: IEEE Symposium on Computational Intelligence and Games (CIG 2010). (2010) 146–153
22. Avery, P., Louis, S.: Coevolving team tactics for a real-time strategy game. In: Proceedings of the 2010 IEEE Congress on Evolutionary Computation (CEC 2010). (2010) 1–8

23. Nogueira, M., Cotta, C., Fernández-Leiva, A.J.: An analysis of hall-of-fame strategies in competitive coevolutionary algorithms for self-learning in rts games. In: Proceedings of the 2013 Learning and Intelligent Optimisation Conference (LION 7). (2013) To appear
24. Authors: Anonymous bot. In: Proceedings. (20XX) pp