Natural Computing manuscript No.
(will be inserted by the editor)

Virtual Player Design using Self-learning via Competitive Coevolutionary Algorithms

Mariela Nogueira Collazo · Carlos Cotta · Antonio J. Fernández-Leiva

Received: date / Accepted: date

Abstract The Google AI Challenge is an international contest the objective of which is to program the Artificial Intelligence (AI) in a two-player Real Time Strategy (RTS) game. This AI is an autonomous computer program that governs the actions that one of the two players executes during the game according to the state of play. The entries are evaluated via a competition mechanism consisting of two-player rounds where each entry is tested against others. This paper describes the use of competitive coevolutionary (CC) algorithms for the automatic generation of winning game strategies in Planet Wars, the RTS game associated with the 2010 contest. Three different versions of a prime algorithm have been tested. Their common nexus is not only the use of a Hall-of-Fame (HoF) to keep note of the winners of past coevolutions but also the employment of an archive of experienced players, termed the Hall-of-Celebrities (HoC), that puts pressure on the optimization process and guides the search to increase the strength of the solutions; their differences come from the periodical updating of the HoF on the basis of quality and diversity metrics. The goal is to optimize the AI by means of a self-learning process guided by coevolutionary search and competitive evaluation. An empirical study on the performance of a number of variants of the proposed algorithms is described and a statistical analysis of the results is conducted. In addition to the attainment of competitive bots we also conclude that the incorporation of the HoC inside the primary algorithm helps to reduce the effects of cycling caused by the use of HoF in CC algorithms.

Keywords Coevolution · Competition · Self-learning · RTS Game · Virtual Player

University of Computer Science
Havana, Cuba
Tel.: +53-835-8845
E-mail: mnogueira@uci.cu

University of Málaga
Málaga , Spain
Tel.: +34-952-13-7158 || -3315
Fax.: +34-952-13-1397
E-mail: {ccottap, afdez}@lcc.uma.es

1 Introduction

The videogame sector represents the largest of the entertainment industries and is an area that is constantly evolving. The main purpose of videogames is to provide entertainment to the player(s) but how to infact do this, is an open question that has yet to be fully answered. In the past, game developers primarily concentrated on having more realistic games and worked on implementing games with high quality graphics (i.e., having high resolution textures, a good measure of frames-per-second, etc.). Whilst, until recently, this policy guaranteed reasonable profits for the development company, in the last decade it has not been enough to ensure the success of a game, as players now demand other features for their games. Many of these required features are associated with the resolution of problems that demand knowledge from a wide set of research domains such as art, psychology, narrative, or music to name but a few.

In this heterogeneous context, one paradigm that is present in all (or in most) videogames is Artificial Intelligence (AI). Currently, it is applied to most aspects of game development and design such as learning, human player imitation, procedural content generation, intelligent camera control, automatic game testing, player/opponent modeling, and computational narrative, among others [24]. The application of AI, and computational intelligence (CI) (as the AI representative of the nature-inspired computational techniques for learning and optimization such as evolutionary algorithms, artificial neural networks, swarm intelligence and ant-colony optimization methods, to give a few examples) in games is thus an active research field that poses significant challenges to the game development community [25]. Nonetheless, the most traditional application of AI in games is to govern the behavior of the non-player-characters (NPCs) in the game with the aim of endowing intelligence on the enemy and as a consequence increasing the satisfaction of the player who demands an opponent who exhibits intelligent behavior. However, generating an AI to control NPCs is a hard task –traditionally tackled by hand– that also requires a quite a large dose of patience. For instance, in triple-A videogames the cost of developing the AI is very high due to the huge number of possible situations in which the NPC might be at a given instant of the game and to the fact that in many commercial games the AI that controls an NPC is still made up of very specialized scripts (previously programmed based on the experience of the designers/programmers and via some traditional AI technique – e.g., finite state machines, of fuzzy logic). These are usually at the root of very well-known problems associated with "artificial stupidity" [22] (e.g. loss of reality, or the predictability and the existence of holes/bugs in the NPC behavior).

It is well-known that a NPC behavior which is too stupid is not desirable as this always leads to a very easy victory for the player thus decreasing his/her satisfaction; an analogous argument can be applied to a very specialized game AI as the player would then most probably suffer a heavy defeat with the same consequence. As an intermediate solution, commercial videogames offer different difficulty levels to the player so that in the most complex levels the player faces high-quality (again pre-programmed) difficult-to-beat opponents. Once the player is able to beat all the opponents in each level, they lose interest. In this context, the generation of opponents whose behavior evolves in accordance with the player's increasing abilities is an appealing feature that makes the game more attractive. In

the literature can be found many proposals for generating NPCs whose behaviors self-adapt to player skills [39,16].

One of the most interesting game research problems is that of developing AI for Real-Time Strategy (RTS) games. In general, in an RTS game the players make decisions in a team (or army) consisting of a number of (not necessarily equal) units (e.g., soldiers, workers, ...) and each player has two main objectives: (a) to destroy the opponent's assets and (b) to create additional structures with some specific goal (e.g., construct buildings to protect their units or defend a specific position in the map). The objectives have to be achieved with an initial limited number of resources because during the game it is possible to obtain more resources. So an RTS game can be viewed as a resource gathering game. Here, game AI means to define the behavior of a virtual player that controls one of the teams in the game. These features and the complexity of the search space (states describe large playing scenarios with hundreds of units simultaneously acting, and the environments consist of many thousands of possible positions for each of the hundreds, possibly thousands, of units) make the design of game AI a very interesting challenge. However this is at the same time, a very hard to handle task due to the specific problems caused by the large search spaces and the parallel nature of the problem – unlike traditional games, any number of moves may be made simultaneously [11].

RTS games provide a range of challenging problems for AI design, e.g., planning in an uncertain world with incomplete information, learning, opponent modeling, and spatial and temporal reasoning [9] just to name a few. One of the most significant challenges is to provide non-cheating and human-like virtual players. To quote Lucas et al. [25], this basically involves "restricting the information available to the AI player to the information a human player may be able to gather in the game, and restricting the actions of the AI player to human player actions (executed in time and space". Additionally, the virtual player should pose a challenge to the human players independently of their skills and of the strategies they adopt to play.

This paper deals with the automatic generation of self-adaptive AI for NPCs in Real Time Strategy (RTS) games. This can be catalogued as a form of procedural content generation (PCG) and combines the advantages of PCG and self-adaptation as for instance level adjustment to player skills and the possibility of producing endless games among others [40]. In particular, we consider coevolution, a biologically inspired technique based on the interaction between different species which in our opinion represents one of the most interesting approaches to be exploited in the evolutionary programming area.

Coevolutionary systems are usually based on two kinds of interactions, akin to symbiotic and predator/prey relationships in nature. The former is a cooperative approach in which an individual is typically decomposed in to different components that evolve simultaneously and the fitness of which depends on the interaction between these components; the latter is a competition-based approach in which an individual competes with other individuals for the fitness value and, if appropriate, will increase its fitness at the expense of its counterparts, whose fitnesses decrease. As in predator/prey relationships in nature, this second approach is prone to trigger "army races" in which the improvement of some individuals stimulates the improvement in the opponents, and vice versa.

This technique has been shown to be successful in a number of applications but it also has a number of drawbacks [15]. Many approaches, e.g. [13], center on the analysis of the dynamics of the coevolutionary process with the aim of identifying both its weaknesses and strengths and to produce more solid techniques for coevolution support. Miconi [26] also underlines the importance of the terms superiority and progress, to avoid fails in the coevolutionary search process. The first remedy to the inherent pathologies of coevolution consisted of proposing some forms of evaluating individuals [34], and the memorization of a number of successful solutions to guide the search. Following this idea, Rosin and Belew [33] proposed the use of a *Hall-of-Fame* (HoF) based mechanism as an archive method. It acts as a long-term memory mechanism in competitive coevolutionary algorithms for managing the historical set of champions during the individuals' evaluation.

The work presented here, builds on our previous work [30,31] on testing the use of HoF-based competitive coevolutionary (HoFCC) algorithms for finding winning strategies in RTS games. In our previous papers we analyzed how the diversity and the growth of the HoF can influence the quality of the solutions obtained by HoFCC algorithms. This was done in the context of the RTS game *RobotWars*[1], a self-developed game the simplicity and inherent limitations of which handicapped the scope of our experimentation; the CC algorithms that imposed certain diversity and tried to maintain a manageable size of the HoF (by removing those champions not contributing to the optimization process) demonstrated a better performance than the others that did not work on the two metrics considered but they still suffered from the appearance of cycling. This is a well-known problem of competitive coevolutionary algorithms that work with 'archive' structures such as the HoF. Now we considerably extend our previous work by considering a new RTS game –namely *Planet Wars*, the *Google AI challenge in 2010*– that allows a deeper experimental analysis and therefore provides more consistent conclusions. We also propose a different evaluation mechanism to exploit the potential offered by archive methods to maintain transitivity between the solutions. Moreover we add novel strength indicators which are independent from the fitness function with the objective of avoiding the appearance of cycling. The novelty of this last aspect consists of incorporating into our prime CC algorithm, an additional archive (termed call-of-celebrities, HoC) that contains a team of experienced virtual players that are used to evaluate how strong a candidate is. The combined use of both halls (HoF and HoC) with the (possibly combined) employment of diversity and quality metrics helps the optimization to obtain competitive bots that self-adapt to beat their (co)evolved enemies.

2 Background

This section discusses a number of approaches that are related to competitive coevolution applied in games (Section 2.1), and describes the RTS game used in our experimental section (Section 2.2).

2.1 (Competitive) Coevolution in games

Due to the intrinsically competitive nature of the games, many researchers have opted for the application of a competitive coevolution approach to solve searching and self-learning problems in games, e.g., the study described in [3] on competitive fitness functions in the Tic Tac Toe game, the application of simple competitive models for evolving strategies in a pursuit-evasion game [32], or the evolution of both morphology and behaviors of artificial creatures through competition in a predator-prey environment [36]. Also [4] analyzed the level of difficulty involved in finding solutions with a basic coevolutionary algorithm for zero-sum games as well as non-zero-sum games; a fitness metric with the score obtained by an agent when faced with another one belonging to the same generation was employed; the authors detected that co-evolving good strategies for zero-sum games is more difficult than for non-zero-sum games.

Competitive coevolution has been used heavily in complex scenarios such as those that emerge in strategy games; so, [37] developed coevolved artificial intelligent opponents with the objective of training human players in the context of a capture-the-flag game. Also, [5] analyzed the use of coevolution for creating a tactical controller for small groups of game entities in a real-time capture-the-flag game; a representation for generating adaptive tactics using coevolved Influence Maps was proposed, and the result was the attainment of an autonomous entity that plays in coordination with the rest of the team to achieve the team objectives. Related with this line of research, [27] presented a spatial decision making system within the context of a 3D computer RTS game, that used a basic implementation of co-evolution in which players were firstly evolved against static hand-coded opponents and later against another population of co-evolving players. More recently, [12] has explored several methods for automatically shaping the coevolutionary process, and this is done by modifying the fitness function as well as the environment during evolution. Another interesting perspective was presented in [8] where the authors, using the game of Tempo as a test space, facilitated the selection of optimal strategies by clustering the solutions in the population of a coevolutionary system through the concept of similarity. This cluster system integrated a long-term memory that valued the changes produced in the environment to trigger appropriate coevolution. The game of Tempo has also been used with the aim of improving the creation of smart agents in [18] and [6].

More closely related to our work, we can cite several articles that address the use of archive methods as an alternative to avoid the effect of pathologies, and that present interesting proposals for optimizing their performance. For example, [29] propose a competitive co-evolution model that introduces the concept of package as a set of good strategies and the best package is the one that contains the smallest number of strategies providing, at the same time, the highest number of victories. Different forms of archives, like the Layered Pareto-Coevolution Archive (LAPCA) [19] and the Coordinate System Archive (COSA) [17] have also been proposed. However, one can find other simpler structures like for instance the use of one simple champion memory that can be improved via different proposals. [21] describes three useful extensions of the HoF that include uniqueness, manual teachers, and Competitive Fitness Sharing [33]; the results showed that HoF works better than SET (Single Elimination Tournament) [3] but this method was not able to prevent the lack of diversity in the population. In [23] the author presents a

system in which different levels of a hierarchical AI coevolve in a simple RTS game environment; archive methods were applied to avoid some of the pathologies of coevolution using fixed, non-evolving opponents during the evaluation process. More recently [35] presents a set of measurements to identify cycling in a population, and proposes an algorithm that minimizes the effect of cycling in a coevolutionary system for the game Othello; in the experiments the authors used different algorithms for generating artificial players, including one that uses a simple archive method with fixed size, which obtained good results.

According to [7], the question of how to actually use the memory in the coevolution tends to be split into two areas: inserting individuals from memory into the coevolution, or evaluating individuals from the populations against the memory. Our investigation falls comes under the latter, and we have implemented different variants of Hall-of-Fame for controlling evaluation process, in a Competitive Coevolutionary (CC) Algorithm.

However, although the contribution of the "archive methods" to reduce the number of drawbacks (e.g., missing features and cycling) of coevolutionary systems is undeniable, their implementation entails a number of issues that have to be taken into account such as the strength of the archive members and the control of the archive diversity. So for instance, a high risk of generating cycles can appear if the strength of a member is valued by pitting this member against a fixed set of opponents, and this is precisely what we did in our previous work [30,31]. A key point to consider is the metric that defines the strength of an individual, which should indicate when an individual is qualitatively superior to the rest but without losing sight of the fact that sometimes quantitative superiority does not correspond with a high strength.

This paper addresses precisely this issue and proposes the consideration of an archive of experts in the evaluation of individuals, in addition to the classic archive of champions. This expert archive will include a fixed set of high quality opponents that will be used to provide an additional evaluation (to that already provided by the use of the champion memory, i.e., the HoF) to measure the quality of the individual; for our particular RTS game and this paper, this expert set is conformed by the collection of built-in bots provided by the *Google AI Challenge 2010* competition plus an optimized genetic algorithm-based bot from [14]. The idea of adding an expert archive for the assessment of candidate fitness is to reduce the likelihood of cycling during coevolution. The fitness of an individual will be calculated by pitting it against its ancestral opponents, but individuals with better fitness will be up against one of the experts' bot (chosen at random) in various scenarios (i.e., different maps); only in case that this individual beats them three times will it be included in the champions' memory. In doing this, we are trying to exploit the potential offered by archive methods to maintain transitivity between the solutions, but adding strength indicators which are independent from the fitness function with the idea of preventing the appearance of cycling.

Another important aspect to consider is the control of diversity to avoid the redundancy in the memory of champions (which may also combat the over-specialization); we also consider this issue here and propose an algorithm (HoFCC-Diversity) that updates the archive from a diversity metric and eliminates those solutions identified as "less diverse". The paper also proposes a multi-objective HoFCC algorithm based on both metrics: strength and diversity, different variants from this algorithm are compared empirically.

Note that this section does not mention other forms of coevolution that have also been applied in games because they are not directly related with competition (e.g., [10] focused on cooperation and not on competition).

2.2 Planet Wars

Planet Wars was the game used in Google Artificial Intelligence Challenge 2010[2], this contest was run by the University of Waterloo Computer Science Club and supported by Google. The game takes place on a map which contains several planets, each of which has a specific number of ships on it. The planets may belong to the player, the opponent, or just be neutral. Each planet owned by a player (not those that are neutral) will increase the forces there according to the "growth rate" of the planet. This rate indicates the ratio of growth to the number of ships on the planet. Figure 1 shows a screenshot of this game, in which the red planets belongs to the player, the green to the enemy, and the rest are neutral.

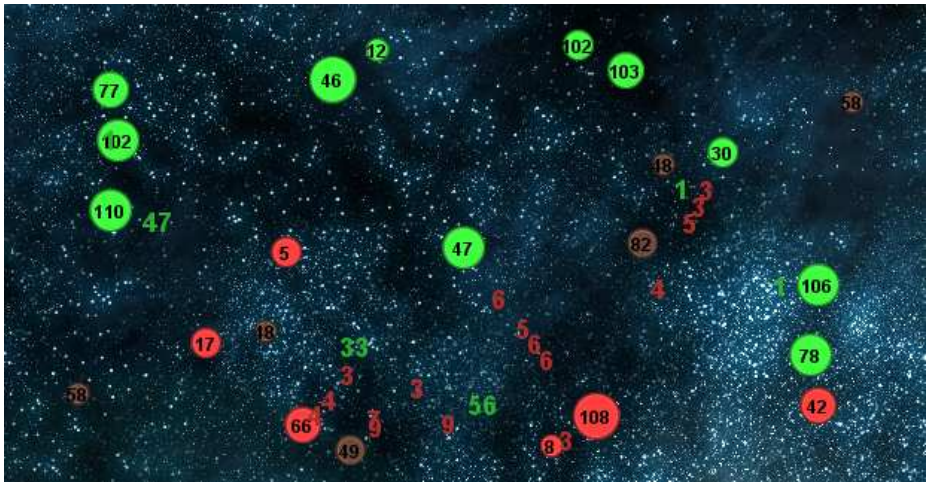

Fig. 1 Screenshot of Planet Wars: Both planets and fleets are identified by a number which shows the number of ships that they have. Their colors also identify the player who owns them.

At the start of each turn, the player receives the current status of the game (i.e., information about the planets and fleets) and can only do one type of action: send fleets of ships from any planet the player owns (i.e., those planets where the number of ships is higher than 0 and the ships belong to the player) to any other planet on the map. Players may send as many fleets as they wish in a single turn as long as they own enough ships. After sending fleets, each planet owned by a player will increase the number of ships staying there proportionally to the planet growth rate. The fleets that were sent in a previous order take a number of turns to reach their destination according to the distance between the planet of origin and the destination one; upon the arrival of the ships if both the planet and the ships belong to the same player, then the number of ships increases by adding up the current number of ships on the planet and the number of newly arrived ships.

Otherwise, if the arrival planet is neutral, then it has a fixed number of ships (NS) and the player must be sent at least $NS+1$ ships to own the planet (i.e. for reaching the neutral planet in order to conquer it); and if the player sends fleets to an enemy planet (i.e. the player attacks an enemy planet) a fight is initiated for the control of the planet (ships from both sides destroy each other until the player with the highest number of ships owns the planet, in this case the destination planet does not have a fixed number of ships, because he/she increments his/her force according his/her growth rate).

Fleets cannot be redirected once they start their journey. Players may continue to send more fleets in later turns even while older fleets are in transit. Although the players issue their orders on a turn-by-turn basis, they carry out these orders at the same time, so we can treat this game as a real-time one.

The player with the most ships at the end of the game wins. The game may also end earlier if one of the players loses all his/her ships and in this case the player that has ships remaining wins instantly; also if one player exceeds the time limit without completing his/her orders she forfeits the game. If both players have the same number of ships when the game ends this is considered a draw.

3 Competitive coevolution for self-learning in Planet Wars

This section describes our competitive coevolutionary approaches. Sect. 3.1 centers on codification issues for the representation of a game strategy, and Section 3.2 details our primary CC algorithm that employs the HoF as a long-term memory mechanism, and the HoC as the expert archive for evaluating the strength of individuals; three variants of this algorithm are also explained.

3.1 Representation issues

Our goal is to find winning strategies, governing a bot (i.e., the virtual player) in Planet Wars by optimizing the rules that guide its decision making so that in each turn the bot should select the best action according to its current state in the game. Here the state at a specific instance of the game is determined by the advantage (with respect to its opponent) of the bot in terms of ships and growth rate. Both advantage metrics have three possible values ('high', 'null' or 'low') which indicate the level of advantage in each case. Lets say that, for player p (resp. opponent o), GR_p (resp. GR_o) is the total "growth rate", according to the "growth rate" of the planets owned by player p, and let $\Delta GR_{po} = GR_p - GR_o$; then if $\Delta GR_{op} > 0$, we say that player p has a 'high' advantage over opponent o in terms of the "growth rate"; 'null' in the case of a draw (i.e., $\Delta GR_{po} = 0$), and 'low' if $\Delta GR_{po} < 0$. The calculation of the advantage in terms of ships is similar but considering NS_p instead of GR_p (resp. NS_o instead of GR_o) as the total number of ships owned by player p (resp. opponent o) at the current instant of the game; in other words, $\Delta NS_{po} = NS_p - NS_o$. However, we considered different thresholds for the advantage values so that if $\Delta NS_{po} > 10$, then player p has a 'high' advantage, 'null' if $0 \leqslant \Delta NS_{po} \leqslant 10$, and 'low' otherwise (note that we consider that a difference of 10 ships is not significant enough to distinguish between the two players; this value was arrived at through our experience of playing).

So, a virtual player strategy is coded as a bidimensional matrix (see Figure 2), where the first dimension symbolizes the player's advantage over his/her opponent in terms of the number of ships (i.e. the ΔNS_{po}), and the second dimension represents the advantage in terms of the total "growth rate" (i.e. the ΔGR_{op}), and as we explained above each axis has three possible values ('high', 'null' or 'low'). Each cell in the matrix acts as a gene and stores one of the next possible actions:

1. attack the strongest enemy planet (AS) (i.e., the enemy planet that owns the highest number of ships),
2. attack the weakest enemy planet (AW),
3. attack the closest enemy planet (AC),
4. conquer the strongest neutral planet (CS) (again in terms of number of ships),
5. conquer the weakest neutral planet (CW),
6. conquer the closest neutral planet (CC),
7. follow the enemy (FE)(in this action the order is to send fleets of ships to the planet to which the enemy is now sending his/her own fleets).

This way, the whole matrix represents a strategy that controls, deterministically, the behavior of a bot during the game by executing the action associated with a specific instance of the game. For a virtual player there are nine possible different states (i.e. 3×3, all the possible value combinations considering the two dimensions of the matrix). And basically, in a specific turn of the game the player will execute the action stored in the state in which he perceives that he is. See Figure 2 for an example of the actions' matrix which shows the distribution of the actions for each possible state, e.g. when the player has a high advantage with respect to his/her opponent in terms of "growth rate" and "ships" ($\Delta GR_{op} > 0$ and $\Delta NS_{op} > 10$) the selected action will be "attack the strongest enemy planet (AS)". Note that the search space is $7^9 = 40353607 \in [2^{25}, 2^{26}]$, which cannot be exhaustively assessed due to the cost of the evaluation that requires a game simulation and thus metaheuristic techniques are used.

	ΔNS*po* > 10	0 <= ΔNS*po* <= 1 0	ΔNS*po* < 0
ΔGR*op* > 0	AS	AC	AW
ΔGR*op* = 0	AS	AC	CW
ΔGR*op* < 0	FE	AS	AW

Fig. 2 A matrix of actions which was generated by our coevolutionary process.

3.2 Hall-of-Fame based Competitive Coevolutionary Algorithm and Variants

Using Planet Wars we propose, in this paper, a modified and improved version of a competitive coevolutionary algorithm that we already described in [30,31] for a simplistic capture-the-flag game. The latter used the HoF as a long-term memory mechanism to keep the winning strategies found in each coevolutionary step and all

of them were also used in the evaluation process (in the basic algorithm). In fact, the best individual from each coevolutionary iteration is retained for future testing, and so we obtain an historical set of champions which is used in the evaluation of the individual. Each population maintains its own HoF, in which its own winning strategies (with respect to the set of winning strategies of its opponent) found in each coevolutionary step will be saved. So, in the coevolutionary step n each possible solution of army A again fights with each solution in $\{B_1, B_2, ...B_{n-1}\}$, where B_i is the champion found by army B in the i-th (for $1 \leqslant i \leqslant n-1$) coevolutionary step.

Regarding the use and implementation of HoF some aspects must be defined. The first is the criteria for inserting a new member in the memory. In this paper we have changed our previous proposal [31] and we have set that an individual must defeat all the HoF members (i.e., all champions of the opponent population), and has to simultaneously beat an expert bot belonging to an archive of efficient (possibly hand-coded) bots; we call this archive *Hall of celebrities* (HoC).

We have then considered different policies for maintaining the champions in the set. For this issue, we take into account the contribution of the individual (i.e., the champion) to the search process as, for instance, it might be the case that some opponents that belong to very old generations do not show a good performance in comparison with opponents generated in recent generations and thus they might be easily beaten; it is therefore crucial to remove those champions that do not contribute to the solution which, in other words, represents a mechanism to control the size of the champions' memory, and is an important element because taking into consideration all the champions (during the evaluation process) might produce more consistent solutions at the expense of a very high computational cost (note that many simulations of the match must be executed for each champion involved in the evaluation; we will provide more details on this further on).

In what follows we present our new proposal for a prime HoF-based competitive coevolutionary algorithm (HoFCC) and three variants of it that differ precisely in the policy of establishing the aspects mentioned previously. From a general point of view, basically the variants differ from each other in the way that they periodically update the solution set on the basis of quality and diversity metrics.

Basic HoFCC

Algorithm 1 shows the schema of our primary algorithm HoFCC (a revised and improved version of that presented in [31]). The algorithm has two parameters: α that indicate (in percentage) the portion of the HoF to be removed with the aim of maintaining only those worthwhile champions that might contribute to the solution according to some specific metric (see details below) and λ the frequency of executing this updating of the HoF.

A specific strategy is considered 'winning' if it achieves a certain score when it deals with the strategies belonging to the set of winning strategies of its opponent (i.e., the enemy Hall-of-Fame), and it defeats an expert bot belonging to the Hall-of-Celebrities. The initial objective is to find a winning strategy of player 1 with respect to player 2 (i.e. the initial opponent) so that the HoF of player 2 is initially loaded with some strategies (randomly or manually initialized: line 2). The hall-of-

celebrities is also loaded with a set of (other demonstrated) efficient virtual players that will be used to evaluate the strength of the solutions (line 2).

The Hall-of-Fame of the player being evolved is updated (i.e., only those robust champions are kept) according to some specific criteria every λ coevolutionary step (lines 5-7). Then a standard evolutionary process tries to find a strategy for player 1 that can be considered victorious (lines 12-17). A strategy is considered winning if its fitness value is above a certain threshold value ϕ (line 18) (which indicates that this strategy has defeated the members of the opponent HoF) and at the same time it has been able to defeat one of the (randomly chosen) celebrities (i.e., 'experts') in three consecutive battles on different maps (i.e., scenarios) of the game; if they are indeed successful, this strategy is added to the HoF of player 1 (line 20) and the process is initiated again but with the player roles reversed (line 21); otherwise (i.e. no winning strategy is found) the search process is restarted. If after a number of coevolutionary steps no winning strategy is found the search is considered to have stagnated and the coevolution ends (See the while condition in line 4). At the end of the entire process we obtain as a result two sets of winning strategies associated accordingly with each of the populations.

Algorithm 1: HoFCC(α, λ)

```
1  nCoev ← 0; A ← player_1; B ← player_2; φ ← thresholdvalue;
2  HoF_A ← ∅; HoF_B ← InitialOpponent(); HoC ← ExpertBotSet();
3  pop ← Evaluate(HoF_B); // Evaluate initial population
4  while nCoev < MaxFailCoevolutions ∧ NOT(timeout) do
5      if nCoev mod λ = 0 then //HoF updating every λ Coevolutions
6          purge_α(HoF_A); // HoF updating
7      end if
8      pop ←RandomSolutions(); // pop randomly initialized
9      i ← 0;
10     foundWinner ← false;
11     e ← RandomElementFrom(HoC);
12     while (i < MaxGen) ∧ ¬foundWinner do
13         parents ←Select (pop);
14         childs ← Recombine (parents, p_X);
15         childs ← Mutate (childs, p_M);
16         pop ← Replace(childs);
17         pop ← Evaluate(HoF_B);
18         if (fitness(best(pop)) ≥ φ) ∧ (beatExpert(best(pop), e)) then //winner found!
19             foundWinner ← true;
20             HoF_A ← HoF_A ∪ {best(pop)};
21             temp ← A; A ← B; B ← temp; // interchange player' roles
22         else
23             e ← RandomElementFrom(HoC);
24             i ← i + 1;
25         end if
26     end while
27     if foundWinner then
28         nCoev ← 0; // start new search
29     else
30         nCoev ← nCoev + 1; // continue search
31     end if
32 end while
```

Regarding the evaluation of candidates for a specific player p (where $p \in \{player\ 1, player\ 2\}$), the fitness of a specific strategy is computed by pitting it against a selected subset of the (winning) strategies in the Hall-of-Fame of its opponent player (that we call the "selected opponent set") and evaluating his qual-

ity in a direct confrontation against any of the expert bots in the hall-of-celebrities. Given a specific strategy s its fitness is computed as follows:

$$fitness(s) = \frac{\sum_{j=1}^{k} \left(r_j^s - nTurn_s(j)\right)}{k} + extras_s(e) \tag{1}$$

where $k \in \mathbb{N}$ is the cardinality of the selected opponent set, $r_j^s \in \mathbb{R}$ returns ϕ points if strategy s beats strategy h_j belonging to the selected opponent set (i.e. victorious case), and 0 if h_j wins over strategy s; $nTurn_{sj} \in \mathbb{N}$ is the number of turns spent on the game (this value is 0 in case of defeat); and $extras_s(e) \in \mathbb{N}$ is a bonus that individual s receives if it defeats the "expert bot" e. This fitness definition was formulated based on our game experience, and it values the victory above any other result.

In the next paragraph we describe in detail the three variants of our HoFCC algorithm which will be tested in the experimental section.

HoFCC-Diversity

In this proposal the HoF acts as a long-term memory mechanism, but the content of the HoF is updated by removing those members that provide less diversity. The value of *diversity* that an individual in the HoF provides is calculated by the genotypic distance as follows: we manage the memory of champions as a matrix in which each row represents a solution and each column a gene (i.e., an action in the strategy). Then, we compute the entropy value for a specific column j as follows:

$$H_j = -\sum_{i=1}^{k} (p_{ij} \log p_{ij}) \tag{2}$$

where p_{ij} is the probability of action i in column j, and k is the memory length. Finally the entropy of the whole set is defined by:

$$H = \sum_{j=1}^{n} H_j \tag{3}$$

The higher the value of H the greater the diversity of the set. For determining the diversity's contribution to a specific solution, we calculate the value of entropy with this solution inside the set, and the corresponding value with this solution outside the set, and finally, the difference of these two values represents the contribution of diversity.

The number of individuals to be deleted from the memory should be set by the programmer as a percentage value (α) representing the portion of the HoF to be removed; in other words, the HoF (with cardinality $\#HoF$) is ordered according to the diversity value in a decreased order and the last $\lceil \frac{\#HoF \times n}{\alpha} \rceil$ individuals in this ordered sequence are removed. The frequency of updating (λ) is also a parameter of this version (i.e., the HoF is updated every λ coevolutions).

The motivation of this proposal is to maintain certain diversity among the members of the HoF, and at the same time reduce (or maintain an acceptable value for) the size of the memory. With this in mind, we assume that the deleted individuals will not affect the quality of the solutions found. Here, the cardinality

of the selected opponent set k in the evaluation phase –see Eq. (1)– is the cardinality of the opponent HoF after executing the updating of the memory (i.e., after removing the individuals).

HoFCC-Quality

In this version, we follow a similar approach to that applied in HoFCC-Diversity but now the HoF is ordered with respect to a measure of quality that is defined as the number of defeats that an individual obtained in the previous coevolutionary step; in other words, a simple counter variable associated with each member of the HoF stores the number of defeats that were computed for the corresponding member during the evaluation process of the opponent army in the previous coevolutionary turn.

Based on our game experience, we assume that this metric is representative of the strength of a solution, and the aim is to keep only the robust individuals in the champions' memory by removing the weak strategies. As in the HoFCC-Diversity, the parameters α and λ have to be set, and the cardinality of the selected opponent set k is exactly the same.

HoFCC-U

This variant of HoFCC follows the idea of optimizing the memory of the champions, but in this case we propose a multiobjective approach where each solution has a diversity value and also a quality value, as previously described, associated with it. Then, a percentage value (α) from the set of dominated solutions according to the multiobjective values is removed; if the set of dominated solutions is empty then HoF is ordered according to the measure of quality and the solutions with worst quality will be removed. As in the previous algorithms (HoFCC-Quality and HoFCC-Diversity) the frequency of updating the HoF is an important parameter that must be defined.

This proposal uses a fitness function different to that shown in Eq. (1) the definition of which was inspired by competitive fitness sharing (CFS) [33]. The main idea is that a defeat against opponent X has more importance if there are other individuals that defeated X.

So, a penalization value N for each individual i (for $1 \leqslant i \leqslant k$) in the population is then calculated as follows:

$$N_i = 1 - \frac{1}{k} \sum_{j=1}^{k} \frac{v_{ij}}{V(j)} \tag{4}$$

where $v_{ij} = 1$ if the i-th individual of the population defeats the j-th strategy (or champion) in the HoF (whose cardinality is k) and 0 otherwise; and

$$V(j) = \sum_{i=1}^{n} v_{ij} \tag{5}$$

is the number of individuals in the population which defeat the j-th opponent of the HoF. As a consequence, $N_i \approx 0$ if the i-th candidate defeats all opponents of

Table 1 Parameters of the coevolutionary cycle

Parameter	Value
$maxFailCoevolutions$	5
$maxCoevolutions$	15
$maxGen$	20
$popSize$	15
$maxEvaluations$	10000
$crossProbability$	$p_X = .75$
$mutProbability$	$p_M = 1/nb$
ϕ	1500
$expertBonus$	500
ω	50
λ	3

HoF and the i-th solution itself is one of the few candidates to do so; $N_i = 1$ if doesn't defeats any opponent; and $0 < N_i < 1$ depending on how many times it wins and how common it is to beat certain opponents. The fitness of a candidate i is then computed as follows:

$$F_i = P_i - \omega N_i \tag{6}$$

where P_i is the result obtained in the battles by Eq. (1), and $\omega \in \mathbb{N}$ is a coefficient that scales N_i in order to make it meaningful with respect to the value P.

4 Experiments and Analysis

This section describes the experimental analysis conducted on the three variants of the HoFCC described in the previous section. All experiments were executed using the Planet Wars engine version 1.2. Five maps from the collection of maps designed for the competition (specifically, map 1, map 10, map 20, map 30, map 50) were considered, so that when an individual faces another one (i.e., its opponent) it must perform three consecutive battles on any of these maps having a maximum of 500 turns for each game. If the individual under evaluation defeats the opponent in all battles then it will be considered the winner.

The initial enemy strategy was defined as 'random' (i.e., we create a random action matrix), and the hall-of-celebrities (i.e., the selection of the "expert bots") comprised the set of bots that were originally provided as "example bots" in the competition (i.e., BullyBot, RandomBot, DualBot, ProspectorBot, RageBot), and specialized hand-coded (and later optimized via evolutionary algorithms) bot (GeneBot) that obtained a ranking position in the top-20% of the Google AI competition 2010 and that was developed by a team from the University of Granada; we selected this bot because it competed efficiently, the source code was available, and it has been extensively described in the scientific literature [14, 28].

Next, we detail the configuration of this experiments, and later will discuss the obtained results.

4.1 Configuration of the Experiments

Nine instances of our algorithms were used: three for each of the HoFCC-Diversity, HoFCC-Quality, and HoFCC-U varying according to the values of $\alpha \in \{10\%, 30\%, 50\%\}$. The notation *HoFX-*$\alpha$ –where $X \in \{Div, Qua, U\}$– is used to denote each of these nine variants. In all cases, we set $\lambda = 3$ and perform 10 runs per algorithm instance, using a steady-state genetic algorithm (GA) –note that this corresponds to Lines 11-16 in Algorithm 1– with the aim of finding a winning strategy with respect to the set of strategies (all strategies stored in the HoF of the opponent) that were considered winning in previous stages of the coevolutionary algorithm; this GA employed binary tournament for selection, uniform crossover, bit-flip mutation, elitist replacement. Table 1 shows the parametrization values, where $maxFailCoevolutions$ represents the limit of continuous coevolutions that one of the players can consume without finding a champion solution; the timeout condition (line 4 in Algorithm 1) is associated with $maxCoevolutions$, which indicates the maximum numbers of total coevolutions that can achieve an algorithm; and $maxEvaluations$ is the limit of evaluations (i.e., $timeout = nCoev > maxCoevolutions \vee numEvaluations > maxEvaluations$, where $numEvaluations$ represents the number of evaluations consumed at that instant by the algorithm). Mutation probability depends on nb which is the number of genes of the individual; ϕ is the threshold value for a winning solution.

Our analysis has been guided by the following indicators which are applied for all runs of each algorithm: *Best fitness*: shows the fitness of the best champion strategy found by the search process; *Average fitness*: shows the average fitness value reached during the coevolutionary cycles; *Number of evaluations*: indicates the total number of battles which are executed during the evaluation process; *Number of victories*: indicates the total number of victories obtained in a *All vs. All* fighting among the best solutions found by each version of algorithm.

Next we analyze the results obtained in ten independent executions for the nine versions of HoFCC, and focus on the indicators mentioned; we have used a non-parametric statistical test to compare ranks namely Kruskal-Wallis test [20] with a significance of 95%. When this test detects significant differences in the distributions, we have performed multiple tests using the Dunn−Sidak method [38] in order to determine which pairs of means are significantly different, and which are not.

4.2 Analysis of the Results

Figure 3 shows the results of the fitness of the best champion strategy found in each independent execution of the algorithm instances. Bear in mind that in our experiments we use three versions of the HoFCC which optimize the use of HoF (in terms of diversity, quality, or both). The Kruskal-Wallis test shows that there are no significant differences between values (See the first row in Table 2). Note however the existence of three outliers (with fitness value 0) in the instances HoFU50, HoFU30 and HoFDiv30; this indicates that in their associated executions no winning strategy was found after completing five consecutive coevolutions.

Figure 4 shows the behavior of the average fitness for each algorithm instance. The Kruskal-Wallis test confirms that the differences between values are statis-

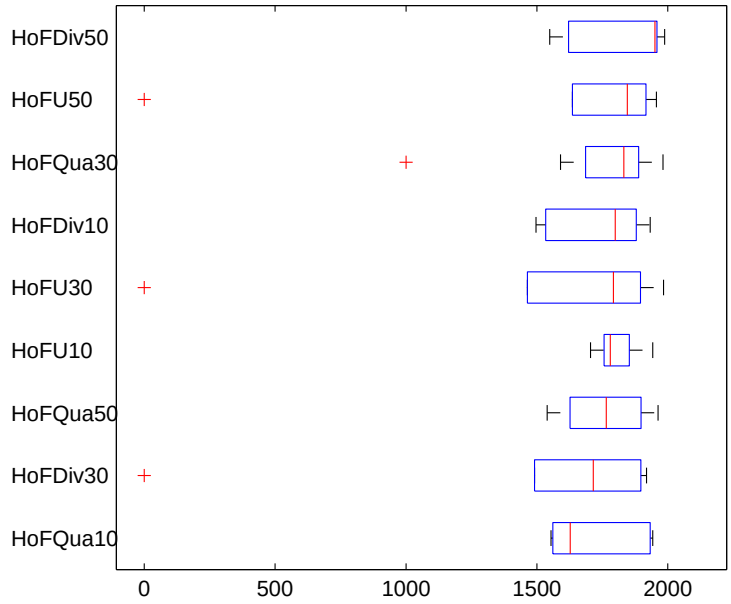

Fig. 3 Distribution of best fitnesses achieved by each algorithm. As usual, each box comprises the second and third quartiles of the distribution, the median is marked with a vertical line, and outliers are indicated with a plus sign.

Table 2 Results of Kruskal-Wallis test for all the indicators.

Indicator	$p-$value
Best fitness	0.4605
Average fitness	$1.356e$-007
Number of evaluations	0.0266
Numbers of victories	0.0093

tically significant (See second row in Table 2). According to the graphic, the algorithms working on 'diversity' obtain the best results. Note also that HoFDiv50 again demonstrates the best performance and that the hybrid versions of the algorithm are not competitive. This last assertion might be made because it is more difficult to obtain a high fitness value due to the penalty that is applied to the score obtained by the individuals according to Eq. (4). A direct consequence is a decrease in the values of the objective function, which on the other hand should not affect the quality of the individual (this will be verified later in another test). In the results of multiple tests for the value of average fitness, the extreme values of the distribution are those that mark the difference: HofDiv50 has significant differences respect to HoFDiv30, HoFU10, HoFU50, HoFU30; and HoFU30 is significantly different from HoFQua10, HoFQua50, HoFDiv10, and HoFDiv50.

In the case of the number of evaluations, there are significant differences (See third row in Table 2) between HoFDiv30 and HoFQua30. We noted however that

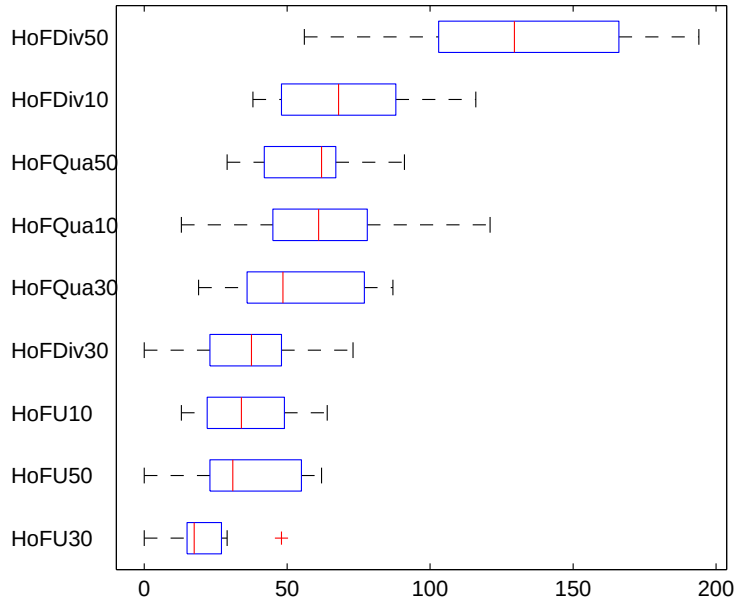

Fig. 4 Distribution of average fitnesses obtained in each algorithm.

the number of evaluations is directly proportional to the length of the coevolutionary cycle; remember that for these experiments we set a maximum quota of 5 consecutive coevolutions without success, and when this limit is reached the algorithm stops. We noted that the lengths of the coevolutionary cycles were not too long, which may be a direct consequence of the specialization of the solutions obtained during the search what makes more difficult the attainment of better solutions.

Ten champions (the best individuals stored in the HoF at the end of the algorithm execution) for each algorithm instance (i.e., one for each execution) were obtained during the experiments and these champions fought in an *All vs. All* tournament (note that in the cases of the outliers with fitness values 0 that were commented previously –regarding best fitness– the undefeated opponent of the HoF was taken as the "winning strategy" in this tournament). This means that every (best) individual competes against the ten champions of the opponent algorithm, and in each confrontation three battles (each of them in a distinct scenario) are executed. Figure 6 shows the results of this tournament. Note how each family of algorithms is clearly distinguished from the others by its results (this could not be perceived in our previous work on the RobotWars game [31]). According to the Kruskal-Wallis test there are significant differences between values (See fourth row in Table 2). The main differences are between HoFDiv and HoFU. Surprisingly all the instances of HoFU version obtain the best results, what means that the algorithms based on the multiobjective approach showed a more efficient behavior, whilst those instances based on the diversity metric exhibited a poor performance

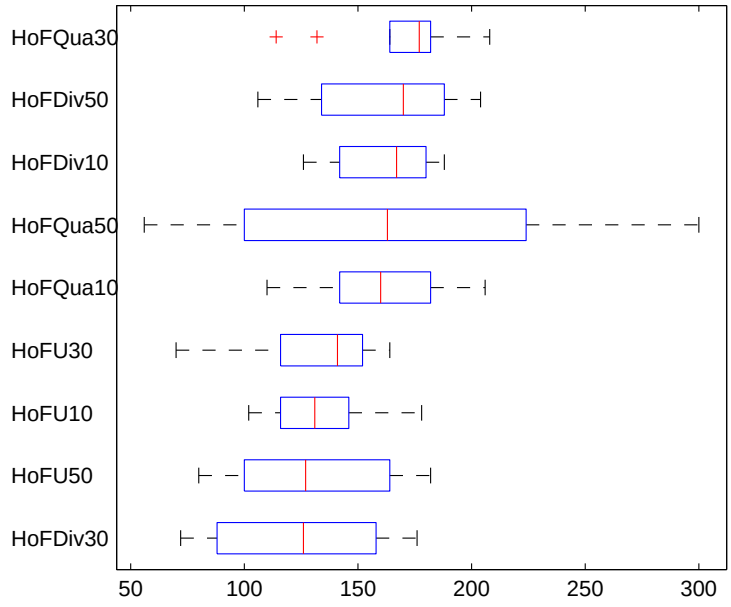

Fig. 5 Distribution of the number of evaluations used by each algorithm.

in direct combats. Certainly, this seems to contradict the results shown in Figures 3 and 4 (See discussion on this in the next subsection) although in fact it is an expected result that confirms that the strongest and more consistent virtual players are those generated from the combined use of the two metrics considered in this paper. We also noted that the quality metric (as expected as well) marks a clear distinction with the diversity measure, although the combination of these is the most productive.

4.3 Summary of Results

We observe that in the All vs. All test, which attempts to measure the strength of the 'champions', the results of the algorithms are not related with the other indicators which focus on the analysis of the fitness values. This may be a sign that the coevolutionary process is affected by cycling and a solution with a high fitness might be theoretically considered as stronger although in practice it might not be; in fact the fitness score is not directly related to the concept of strength (or efficiency in combat). As already mentioned, the strongest virtual players (according to the test *All vs. All*) were obtained by the family of algorithms that use a multi-objective approach based on diversity and quality values as metric to guide the search whilst the algorithms based on just one metric (specially those considering the diversity values) did not exhibit good results. The lowest values associated with the bots generated from the multi-objective instances might be

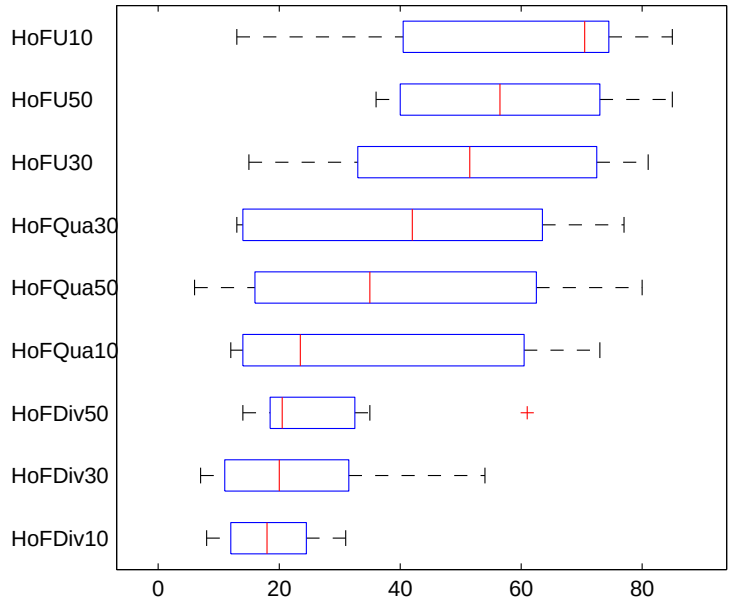

Fig. 6 Distribution of the number of victories obtained by each algorithm in the *All vs. All* fighting tournament

caused by the penalty that is applied to the score in the fitness function according to Eq. (4). To relate all the algorithm instances, in the future we might think to include some of the champions obtained by each of the algorithms in the hall-of-celebrities of the rest of algorithms so that more pressure will be given to the attainment of champions and the fitness value of the individual might be more consistent with their performance.

The results obtained in this paper provide conclusive evidence in favor of the hypotheses set forth in previous work on the RobotWars game [31]. Related to this, note that we include more battles against the members of the rival HoF in the candidate evaluation, and second (and most importantly) we believe that the application of the concept of Competitive Fitness Sharing (CFS) [33] to the fitness function, and the addition of pressure to the search via the inclusion of an expert archive caused a reduction in the appearance of cycling. In this regard, in our previous proposals we assumed that there would be transitivity between solutions and at the last solution added to the archive had to be the strongest, so the solutions were evaluated only by the score that they obtained against the members of the opponent HoF which favored the appearance of cycling and disengaged the search targets. The idea of including in the fitness functions combats against a group of experts (with which they have no evolutionary relationship) helped to break relationships between the champions found in each generation, and this surely eases the cycle breaking.

Regarding the percentage of updating the HoF (ie, 10%, 30%, or 50%), no significant difference was observed because their behaviors were indistinguishable. In this case, we believe that it is due to the fact that the HoF does not reach a significant size because in the experiments we set a limit of total coevolutions, so the HoF cannot grow significantly and it does not allow the capabilities of the updating process to be exploited.

5 Conclusions

Finding algorithms that can reduce the effect of the inherent pathologies of the coevolutionary models is an open research line. In previous work we explored the use of an archive method, in particular the use of a hall-of-fame (HoF), to keep the champions obtained in each coevolutionary step and that was used in the evaluation process to guide the search. An empirical study conducted on a real-time strategy (RTS) game –with intrinsic underlying limitations due to its simplicity– showed that the well-known problem of cycling continued appearing due to the assumption of a transitivity among the champions. Now, in this paper, we have extended that work significantly by introducing changes in the coevolutionary process, trying to promote those solutions that are truly strong. The proposed approach maintains the use of the HoF but also incorporates an additional memory (i.e., another archive termed hall-of-celebrities, HoC) to contain other efficient bots (i.e., 'experienced' or optimized virtual players) that are used here to asses the strength of the solution candidates. This concept of HoC allows pressure to be put on the algorithm search as the expert bots can (and should) be implemented independently in other contexts. This means that these optimized virtual players do not necessarily maintain an evolutionary relationship with the evolutionary population. Moreover, in the experimental analysis we have considered a new RTS game that enables a deeper experimentation and thus the attainment of more consistent results to draw general conclusions. In addition, we have considered two quality metrics and, based on these, we have suggested a number of variants (i.e., families) of a primary CC algorithm that differ in the mechanism of updating the HoF; this updating is executed with the aim of removing those champions not contributing to the optimization. A new fitness function, inspired by the Competitive Fitness Sharing (CFS) [33], was also used in a multi-objective version of the primary CC algorithm.

The results obtained in the experiments clearly distinguish the performance of each algorithm family in the search for strong solutions. In this regard, our multi-objective CC version shows a consistent performance. This multi-objective variant is guided by the quality of the solution candidate (with respect to both HoF and HoC) and the diversity of the Hall-of-fame. Taking into account these two metrics independently, the algorithmic versions led by the quality measure provides better (in the sense of "more competitive") bots than those variants guided by the diversity metric.

An experimental analysis has also shown that our HoF/HoC-based proposal helps to reduce the occurrence of cycling in the coevolutionary process. Moreover, this research have also highlighted the self-adjustment capabilities of the CC algorithms described here by generating winning strategies with respect to both the (co)evolved enemies and other optimized enemies that form part of an experi-

enced line-up. This opens up multiple application in the arena of videogames such as their use on adaptive games [39] among others.

Future work will involve analyzing the performance of new coevolutionary models and the application of those described here on other RTS games, incorporating new metrics for the HoF updating process, trying to design new evaluation mechanisms with the aim of reducing the effects of coevolutionary pathologies, and exploiting the potential of archive methods when they are combined with other approaches which help to optimize the performance of the solutions.

Acknowledgements This work is partially supported by Spanish MICINN under project ANYSELF (TIN2011-28627-C04-01)[3], by Junta de Andalucía under project P10-TIC-6083 (DNEMESIS)[4] and by Universidad de Málaga, Campus de Excelencia Internacional Andalucía Tech.

Notes

[1] `http://wp.me/p2c0bl-60`
[2] `http://planetwars.aichallenge.org`
[3] `http://anyself.wordpress.com/`
[4] `http://dnemesis.lcc.uma.es/wordpress/`

References

1. Proceedings of the IEEE Congress on Evolutionary Computation, CEC 2010, Barcelona, Spain, 18-23 July 2010. IEEE (2010)
2. IEEE Congress on Evolutionary Computation, CEC 2011, New Orleans, LA, USA, 5-8 June, 2011. IEEE (2011)
3. Angeline, P.J., Pollack, J.B.: Competitive Environments Evolve Better Solutions for Complex Tasks. In: S. Forrest (ed.) 5th International Conference on Genetic Algorithms (ICGA93), pp. 264–270. Morgan Kaufmann (1993)
4. Ashlock, D., Ashlock, W., Samothrakis, S., Lucas, S.M., Lee, C.: From competition to cooperation: Co-evolution in a rewards continuum. In: IEEE Conference on Computational Intelligence and Games (CIG 2012), pp. 33–40. IEEE (2012)
5. Avery, P., Louis, S.J.: Coevolving team tactics for a real-time strategy game. In: IEEE Congress on Evolutionary Computation (CEC 2010) [1], pp. 1–8
6. Avery, P., et al.: Coevolving a computer player for resource allocation games: using the game of Tempo as a test space. Ph.D. thesis, School of Computer Science University of Adelaide (2008)
7. Avery, P.M., Greenwood, G.W., Michalewicz, Z.: Coevolving strategic intelligence. In: IEEE Congress on Evolutionary Computation (CEC 2008), pp. 3523–3530. IEEE (2008)
8. Avery, P.M., Michalewicz, Z.: Static experts and dynamic enemies in coevolutionary games. In: IEEE Congress on Evolutionary Computation (CEC 2007), pp. 4035–4042. IEEE (2007)
9. Buro, M.: Call for AI research in RTS games. In: D. Fu, J. Orkin (eds.) AAAI workshop on Challenges in Game AI, pp. 139–141. San Jose (2004)
10. Cook, M., Colton, S., Gow, J.: Initial Results from Co-operative Co-evolution for Automated Platformer Design. In: C.D. Chio, A. Agapitos, S. Cagnoni, C. Cotta, F.F. de Vega, G.A.D. Caro, R. Drechsler, A. Ekárt, A.I. Esparcia-Alcázar, M. Farooq, W.B. Langdon, J.J.M. Guervós, M. Preuss, H. Richter, S. Silva, A. Simões, G. Squillero, E. Tarantino, A. Tettamanzi, J. Togelius, N. Urquhart, S. Uyar, G.N. Yannakakis (eds.) Applications of Evolutionary Computation - EvoApplications (EvoGAMES), *Lecture Notes in Computer Science*, vol. 7248, pp. 194–203. Springer (2012)
11. Corruble, V., Madeira, C.A.G., Ramalho, G.: Steps toward Building of a Good AI for Complex Wargame-Type Simulation Games. In: Q.H. Mehdi, N.E. Gough (eds.) 3rd International Conference on Intelligent Games and Simulation (GAME-ON 2002), pp. 155–159. London, UK (2002)

12. Dziuk, A., Miikkulainen, R.: Creating intelligent agents through shaping of coevolution. In: IEEE Congress on Evolutionary Computation (CEC 2011) [2], pp. 1077–1083
13. Ebner, M., Watson, R.A., Alexander, J.: Coevolutionary Dynamics of Interacting Species. In: C.D. Chio, S. Cagnoni, C. Cotta, M. Ebner, A. Ekárt, A. Esparcia-Alcázar, C.K. Goh, J.J.M. Guervós, F. Neri, M. Preuss, J. Togelius, G.N. Yannakakis (eds.) Applications of Evolutionary Computation, EvoApplicatons 2010 (EvoApplications (1)), *Lecture Notes in Computer Science*, vol. 6024, pp. 1–10. Springer (2010)
14. Fernández-Ares, A., Mora, A.M., Guervós, J.J.M., García-Sánchez, P., Fernandes, C.: Optimizing player behavior in a real-time strategy game using evolutionary algorithms. In: IEEE Congress on Evolutionary Computation [2], pp. 2017–2024
15. Ficici, S.G., Bucci, A.: Advanced tutorial on coevolution. In: 2007 GECCO Conference Companion on Genetic and Evolutionary Computation, pp. 3172–3204. ACM, New York, USA (2007)
16. Gutiérrez, J.A.G., Cotta, C., Fernández Leiva, A.J.: Design of Emergent and Adaptive Virtual Players in a War RTS Game. In: J.M. Ferrández, J.R.Á. Sánchez, F. de la Paz, F.J. Toledo (eds.) 4th International Work-Conference on the Interplay Between Natural and Artificial Computation (IWINAC 2011), *Lecture Notes in Computer Science*, vol. 6686, pp. 372–382. Springer, La Palma, Canary Islands, Spain (2011)
17. Jaskowski, W., Krawiec, K.: Coordinate system archive for coevolution. In: IEEE Congress on Evolutionary Computation CEC 2010 [1], pp. 1–10
18. Johnson, R., Melich, M., Michalewicz, Z., Schmidt, M.: Coevolutionary Tempo game. In: Congress on Evolutionary Computation (CEC 2004), vol. 2, pp. 1610–1617 (2004)
19. de Jong, E.: Towards a bounded Pareto-Coevolution archive. In: Congress on Evolutionary Computation (CEC2004.), vol. 2, pp. 2341–2348. IEEE, Portland, Oregon, USA (2004)
20. Kruskal, W., Wallis, W.: Use of ranks in one-criterion variance analysis. Journal of the American Statistical Association **47**(260), 583–621 (1952)
21. Lichocki, P.: Evolving players for a real-time strategy game using gene expression programming (2008). Master thesis, Poznan Universtity of Technology
22. Lidén, L.: Artificial Stupidity: The Art of Intentional Mistakes. In: AI Game Programming Wisdom 2, pp. 41–48. Charles River Media, INC. (2004)
23. Livingstone, D.: Coevolution in hierarchical ai for strategy games. In: IEEE Conference on Computational Intelligence and Games (CIG 2005). IEEE (2005)
24. Lucas, S.M.: Computational Intelligence and AI in Games: A New IEEE Transactions. IEEE Trans. Comput. Intellig. and AI in Games **1**(1), 1–3 (2009)
25. Lucas, S.M., Mateas, M., Preuss, M., Spronck, P., Togelius, J.: Artificial and Computational Intelligence in Games (Dagstuhl Seminar 12191). Dagstuhl Reports **2**(5), 43–70 (2012)
26. Miconi, T.: Why Coevolution Doesn't "Work": Superiority and Progress in Coevolution. In: L. Vanneschi, S. Gustafson, A. Moraglio, I.D. Falco, M. Ebner (eds.) Genetic Programming, 12th European Conference (EuroGP 2009), *Lecture Notes in Computer Science*, vol. 5481, pp. 49–60. Springer (2009)
27. Miles, C., Louis, S.J.: Co-evolving real-time strategy game playing influence map trees with genetic algorithms. In: International Congress on Evolutionary Computation (CEC 2006). IEEE, Vancouver, Canada (2006)
28. Mora, A.M., Fernández-Ares, A., Guervós, J.J.M., García-Sánchez, P., Fernandes, C.M.: Effect of Noisy Fitness in Real-Time Strategy Games Player Behaviour Optimisation Using Evolutionary Algorithms. J. Comput. Sci. Technol. **27**(5), 1007–1023 (2012)
29. Nerome, M., Yamada, K., Endo, S., Miyagi, H.: Competitive co-evolution based game-strategy acquisition with the packaging. In: L.C. Jain, R.K. Jain (eds.) Knowledge-Based Intelligent Electronic Systems, 2nd International Conference (KES (3)), pp. 184–189. IEEE (1998)
30. Nogueira, M., Gálvez, J., Cotta, C., Fernández-Leiva, A.: Hall of Fame based competitive coevolutionary algorithms for optimizing opponent strategies in a new RTS game. In: A. Fernández-Leiva et al. (ed.) 13th annual European conference on simulation and AI in computer games (GAME-ON 2012), pp. 71–78. Eurosis,, Málaga, Spain (2012)
31. Nogueira, M., Gálvez, J., Cotta, C., Fernández-Leiva, A.: An Analysis of Hall-of-Fame Strategies in Competitive Coevolutionary Algorithms for Self-Learning in RTS Games. In: International Conference on Learning and Intelligente Optimization (LION7), LNCS. Springer, Catania, Italy (2013). In Press.
32. Reynolds, C.: Competition, coevolution and the game of tag. In: Brooks, P. Maes (eds.) Proceedings of Artificial Life IV, pp. 59–69. MIT Press, Cambridge, Massachusetts (1994)

33. Rosin, C., Belew, R.: New methods for competitive coevolution. Evolutionary Computation **5**(1), 1–29 (1997)
34. Rosin, C.D., Belew, R.K.: Methods for Competitive Co-Evolution: Finding Opponents Worth Beating. In: L.J. Eshelman (ed.) 6th International Conference on Genetic Algorithms (ICGA), pp. 373–381. Morgan Kaufmann (1995)
35. Samothrakis, S., Lucas, S.M., Runarsson, T.P., Robles, D.: Coevolving Game-Playing Agents: Measuring Performance and Intransitivities. IEEE Trans. Evolutionary Computation **17**(2), 213–226 (2013)
36. Sims, K.: Evolving 3D Morphology and Behavior by Competition. Artificial Life **1**(4), 353–372 (1994)
37. Smith, G., Avery, P., Houmanfar, R., Louis, S.J.: Using co-evolved RTS opponents to teach spatial tactics. In: G.N. Yannakakis, J. Togelius (eds.) IEEE Conference on Computational Intelligence and Games (CIG 2010), pp. 146–153. IEEE (2010)
38. Sokal Robert, R., Rohlf James, F.: Biometry: the principles and practice of statistics in biological reseach. W.H. Freeman and Company, New York (1995)
39. Szita, I., Ponsen, M.J.V., Spronck, P.: Effective and Diverse Adaptive Game AI. IEEE Trans. Comput. Intellig. and AI in Games **1**(1), 16–27 (2009)
40. Togelius, J., Yannakakis, G.N., Stanley, K.O., Browne, C.: Search-Based Procedural Content Generation: A Taxonomy and Survey. IEEE Trans. Comput. Intellig. and AI in Games **3**(3), 172–186 (2011)