Rolling Horizon Co-evolution in Two-player General Video Game Playing

Charles Ringer[*][1], Cristiana Pacheco[‡][1], Georgiana Cristina Dobre[†], Diego Perez-Liebana[‡]
[*]University of York
[‡]Queen Mary University of London
[†]Goldsmiths, University of London
cr1116@york.ac.uk, c.pacheco@qmul.ac.uk, gdobr001@gold.ac.uk, diego.perez@qmul.ac.uk

Abstract. Artificial Intelligence for General Video Game Playing (GVGP) is challenging not only because agents must adapt to a range of different games, but they must also make decisions within the time constraints of real-time video games. The General Video Game Artificial Intelligence framework (GVGAI) is a popular framework for GVGP. It features a two-player track where two agents play a game together, either competitively or cooperatively, which poses the additional challenge of considering another player. Commonly, agents only consider their own moves in these two-player games. In this paper we discuss and assess Rolling Horizon Co-evolutionary Planning (a modification to Rolling Horizon Evolutionary Algorithms) for two player GVGAI. We present experimental results on its effectiveness against other agents playing GVGAI games and show that co-evolution can improve results compared to a RHEA agent.

Introduction

Two-player General Video Game Playing (GVGP) is an interesting challenge for AI. This is because agents must be able to form effective plans for a wide range of games, be capable of developing plans expediently (given the real-time nature of these games) and consider the opposing agent in the environment. The second agent's unknown actions add stochasticity to the environment and may impact the player's plans. Thus two-player games can pose a higher challenge for state-of-the-art GVGP agents compared to single-player games.

One competition for developing AI for general video game playing is the General Video Game AI (GVGAI) competition [20]. GVGAI has a large library of single-player and multi-player games that can be used to test agents. Many of these games are re-implementations of classic games, such as *Frogger*, *Pac-Man* and *Sokoban*, along with novel designs. It holds yearly competitions for GVGP across many tracks, including the two-player games track from which we used the games and framework for this study.

The current state-of-the-art in AI agents for two-player GVGAI often uses a random model to simulate the opponent's moves. However, modelling the opponent could be useful when planning actions as the utility of certain action sequences may be highly dependent on the opponent's moves [9]. The main contribution of this paper is an evaluation of Rolling Horizon Co-evolutionary Planning (RHCP) for GVGAI. RHCP is a modification of Rolling Horizon Evolutionary Algorithms (RHEA) where the agent evolves a plan for itself, which represents the moves it may take in the game, alongside a "best guess" plan for the opponent. This method assumes that the opponent is playing rationally, not randomly, and is seeking to maximise its score.

In this paper, the GVGAI framework and its two-player track are briefly discussed along with two existing GVGP approaches: Monte Carlo Tree-Search (MCTS) and RHEA. Next, we detail the RHCP algorithm before presenting a comparison between the algorithm and the previous mentioned approaches, followed by an analysis of the prediction accuracy over them. Finally, we discuss these result and conclude that RHCP is a promising approach to two-player GVGP.

GVGAI

Competitions have been widely used to test game playing AI algorithms. These range from board games (*Go* [23]), platformers (*Super Mario Bros* [24]) and real-time strategy games (*StarCraft* [15]). However, for these, the AI developed will be tailored to one game only and rarely applicable to other games. Thus, the General Game Playing competition was developed [8] where submitted agents are evaluated on unknown (to both the agents and the competitors) turn-based discrete games. Subsequently, GVGAI was developed to test the adaptability of a single AI to many real-time games. Similar environments exist, such as the Arcade Learning Environment [1] and Open AI Gym [2], although these are typically used in iterative learning tasks, not using a forward model.

Framework and Competition

The GVGAI [19] framework uses the Video Game Description Language (VGDL) [22] to describe games and levels, and provides a Forward Model (FM) to the agents. This provides the ability to roll the state forward, simulating possible upcoming states. Sprites can be created and parameterized for a 2D game environment using text files. Currently, it has more than a hundred publicly available single player games and 80 two-player games. In these games, agents receive data about their status in a game state. This includes: score, time step, current winner/loser (if any), sprite positions, orientations, resources and collisions with other sprites. Given this data, it is up to the agent(s) to decipher the games' mechanics and rules. A complete description of implementation and rules can be found at [20]; and examples can be seen in Figure 1. At present there are several tracks

[1] These authors contributed equally.

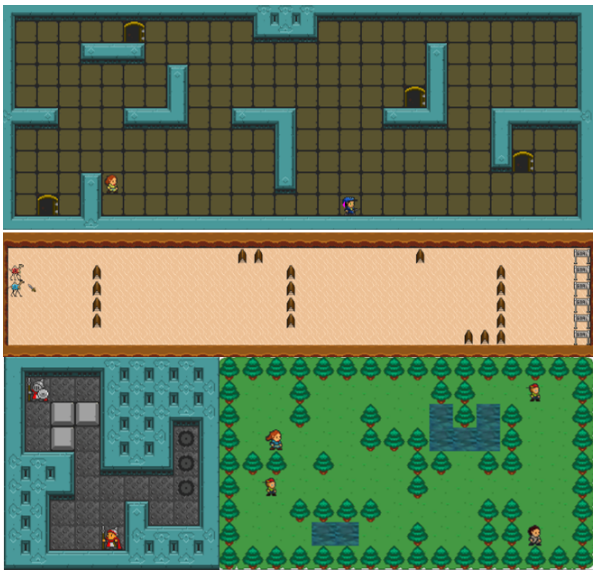

Figure 1. 2D arcade style games from GVGAI: Samaritan (top), Steeplechase (middle), Sokoban (bottom left) and Gotcha (bottom right).

for this competition: single-player learning [25], single-player planning [20], level generation [11], rule generation [10] and the two-player planning [4]. The two-player planning track is the focus on this work.

Two-Player Track

The two-player track extends GVGAI to multiplayer games [4] - by giving the number of players and all game state data to both. Agents can both win, both lose, or only one of them win.

As previously stated, the avatars have access to a Forward Model (FM), allowing them to simulate potential future states in their thinking time. The FM allows agents to make copies of the states and to roll them forward by providing an action for every player in the game. Moves in two-player GVGAI are simultaneous in all games. The agents have no previous information about the game rules or mechanics, hence these can only be estimated given the interactions with the environment and the other player. Games for this track feature both cooperative and competitive scenarios, a detail not revealed to the playing agents. Since these new mechanics are introduced, there are many differences in the games: more interactions are available, score systems and end conditions - such as defence of objects or both players finishing in a given location.

A recent survey in GVGAI can be found in [18], which highlights the different approaches that have been tried in this domain. The two most popular are variants of Monte-Carlo Tree Search (MCTS) [3] and Rolling Horizon Evolutionary Algorithms (RHEA) [16].

Monte Carlo Tree Search

MCTS based techniques are widely used in the GVGAI framework for both single and two-players games. It combines Tree Search - by building said search incrementally - with Monte Carlo simulations - through the evaluation of leaf nodes using the simulation. A multi-armed bandit algorithm is used to balance exploring tree nodes that have little information with exploiting nodes that are known to be good. At each iteration, MCTS performs 4 steps: selection of a node using Upper Confidence Bounds [13] (UCB), expansion of its children, simulation using a random sequence of moves (rollout) and back-propagation of the reward, through the tree, and updating the parent nodes. This process repeats until a predefined computational budget is reached. For a comprehensive overview of MCTS see [3].

Of particular interest to this study, the effects of using an opponent model for MCTS in two-player GVGAI is investigated in [9]. Some of the models implemented are variations of the sample MCTS algorithm, while others follow a probabilistic approach. The latter involves offline learning to have a probability distribution over the possible player's actions on a game state. The results show that the probabilistic models perform the highest win rates. It is interesting to observe that this work is, to the knowledge of the authors, the only one in which an attempt to modelling the opponent is made - all the other approaches for two-player GVGAI assume a random move for the opponent or a similarly simple approach.

Rolling Horizon Evolutionary Algorithms

RHEA are statistical planning algorithms which aim to evolve a sequence of actions (a plan), which maximises a reward function. It was first proposed for real-time games in [16], where the authors applied it to the Physical Travelling Salesman Problem, showing better performance than MCTS in this scenario. Their applications in GVGP has recently been subject to attention, by first studying the vanilla algorithm and its parameters [4], seeding techniques [5] and overall improvements regarding shift buffer, statistical trees and added rollouts [7]. Initially, a population of uniformly random action sequences is generated. Each action is represented by an integer $[0, N-1]$, where N is the number of actions available to the agent. Once this initial population is generated, individuals are evaluated by stepping through the forward model and applying each action in the sequence when the AI is required to act. After this, an heuristic function is used to calculate the value of the game state, which then becomes the fitness for this individual. Once the initial population has been evaluated, genetic operators are applied to evolve a new population of action plans which are, in turn, evaluated and evolved until a set computational budget is exhausted. At this point, the first move in the best individual is returned as the action to take.

Rolling Horizon Co-evolutionary Planning

RHCP is a modification of RHEA where the agents' plans are co-evolved with a population of opponent plans. It was first proposed in [14] although, in this work, experimentation was carried out on only one game. As such, RHCP's viability for GVGP was untested. Whilst the player's plan represents the moves the player will make, the opponent's plans represent "best guess" plans about what actions they may take. This is intended to give the agent some information about strong actions the opponent might take when evolving its own plan. This population of opponent plans is then used when calculating the fitness of individuals in the player's plan population and, likewise, the player plan population is used when assessing the fitness of the opponent's plan. Compared to [9], this approach does not require offline learning of move probability distributions because plans are evolved online, alongside the agent's plan. However, it does assume that the opponent is playing rationally and its moves are similar to those that are discoverable by a RHEA agent.

In the work presented in this paper, two different evolutionary strategies are used depending on whether the agent's plan or the op-

ponent's plan is being evolved. This is because part of the computational budget is dedicated to investigate which moves the opponent may take, but there needs to be a balance with the time spent on planning the own agent's move. It is possible that, for some games, the opponent's moves are not relevant - a racing game where blocking is not possible. Ideally, little computational time should be employed in that case.

Agent move planning: μ + λ Evolutionary Strategy

The population of a player's plan is evolved using a μ + λ evolutionary strategy. First, the population is sorted based on fitness, and elitism is applied, selecting the μ most fit individuals. Then λ individuals are generated by first selecting some individuals from the previous population, using tournament selection of size t_size, and then applying uniform crossover and random mutation to these individuals. Uniform crossover is carried out by, for each gene in the new individual, selecting with equal probability either the gene from one of the parents. Random mutation is carried out by selecting a gene uniformly at random and then setting it to a new valid value, which in this case is an integer [0, $N-1$]. The fitness for this new population is then calculated and this process is repeated until a computational budget is exhausted at which point, the individual with the highest fitness is then returned as the best solution.

Opponent move planning: (1+1) Random Mutation Hill Climber

A (1+1) Random Mutation Hill Climber (RMHC) is an evolutionary algorithm whereby a population of $N = 1$ is evolved by first assessing the fitness of the single individual; followed by mutating its chromosomes using genetic operators and assessing the mutated fitness. If the mutated individual has the same or better fitness, it replaces the original individual as the single member of the population. This process is repeated until a computational budget is exhausted, at which point, the first move in the individual is returned. For this algorithm, the mutation operator used is the same as in the μ + λ ES described above, although it is applied M times to maximize exploration.

State Evaluation

Both the $\mu + \lambda$ ES and $(1 + 1)$ RMHC use the same scoring heuristic when determining the fitness of an individual. The fitness is defined as the game score for the player in the state reached at the end of the sequence, plus a large negative value if the player lost (-1000) or a large positive if the player won (1000). This scoring heuristic is also used in our experiment for the other agents which require a scoring function (RHEA and MCTS).

Co-Evolution

Co-Evolution refers to evolving chromosomes for two (or more) different populations in parallel by using one to influence the fitness of the other [21]. This study presents a methodology for co-evolution of the agent's plan with a "best guess" plan for the opponent. When evaluating the fitness of the agent's plan, we use the best guess opponent plan to simulate what actions the opponent may take. Likewise, when evaluating the opponent's plan, we simulate the actions the agent takes using its best known plan.

In this paper, we use a μ + λ evolutionary strategy to evolve the agent's plan and a (1+1) RMHC to evolve our "best-guess" opponent's plan. In each iteration of the algorithm, the population of plans is evaluated, the elite selected and the rest mutated. Then the opponent plan is mutated and both the original and mutated plans are evaluated against the best known player plan. The opponent plan with the highest fitness is then preserved and survives into the next iteration of the algorithm. This process is repeated until the budget is exhausted. Algorithm 1 details the core RHCP loop, initialisation and evaluation policies; algorithm 2 details the genetic operators for RHCP.

Algorithm 1 Rolling Horizon Co-evolutionary Planning

Requires state: current game state.
procedure ACT($state$)
    init()
    while $budgetremains$ **do**
        evaluate(population, opponent, state)
        evolve(population, opponent, state)
    return $population.best().first\_move()$
procedure INIT
    $population \leftarrow randompopulation$
    $opponent \leftarrow randomindividual$
procedure EVALUATE($population$, opp, $state$)
    for indv in population **do**
        $indv.fit \leftarrow simulate\_game(indv, opp, state)$
    $population \leftarrow sort(population)$
procedure SIMULATE_GAME($indv$, $opponent$, $state$)
    for i in indv.length **do**
        $move_a \leftarrow indv[i]$
        $move_b \leftarrow opponent[i]$
        $state.advance(move_a, move_b)$
    return $score\_heuristic(indv, state)$

Algorithm 2 Rolling Horizon Coevolutionary Planning Genetic Operators

Requires elitism (μ)
procedure EVOLVE($population$, $opponent$, $state$)
    $newPop \leftarrow emptypopulation$
    while $newPop.size < elitism$ **do**
        $newPop \leftarrow add(next\_fittest\_individual)$
    while $newPop.size < population.size$ **do**
        $newPop \leftarrow selectandmutate(population)$
    $mutOp \leftarrow mutate(opponent)$
    $opFit \leftarrow simulate\_game(opponent, bestindv, state)$
    $opMFit \leftarrow simulate\_game(mutOp, bestindv, state)$
    if $opMFit <= opFit$ **then**
        $opponent \leftarrow mutationOpponent$
Requires t_size: tournament size
procedure SELECTANDMUTATE($population$)
    while $tournament.size <$t_size **do**
        $tournament \leftarrow population.rand\_indv()$
    $parentA \leftarrow tournament.fittest()$
    $parentB \leftarrow tournament.secondFittest()$
    $new\_indv \leftarrow uniformCrossover(parentA, parentB)$
    return $mutate(new\_indv, mutation)$
Requires M: mutation repetitions
procedure MUTATE($individual$)
    for M **do**
        $gene_idx \leftarrow individual.new\_random\_gene()$
        $individual[gene_idx] \leftarrow new\_random\_gene()$
    return $individual$

Experiment

For this experiment, 10 games from the GVGAI Two-Player track were used, those found in the GVGAI Two-Player Training Set 1. This training set contains a mixture of Competitive games (8), where there is at most one winner, and Cooperative games (2), where players are working towards a shared goal and win or lose together. Additionally, there is a mixture of Symmetric games(7), where each player has the same goal, and Asymmetric games (3), where each player has a different goal. Finally there is a mixture of Stochastic games (4), those with a random element, and Deterministic games (6), those that have no randomness from the environment. A list of games can be found in Table 1.

Four algorithms were tested: our RHCP agent and 3 control agents. All control agents, RHEA, MCTS, and Random, were modifications of sample agents provided with the GVGAI framework. RHEA and RHCP use both the same length for their individuals ($l = 15$), tournament selection ($t\_size = 2$), uniform crossover and mutation. The only difference between the two algorithms is that RHEA used a population of $n = 10$ whereas RHCP used a population of $n = 8$. This was to allow computational budget for the secondary population for the opponent's plan, such that it also evaluates 10 individuals per generation. In RHCP, the player's plan population has a μ of 1, selected through elitism, and $\lambda = n - \mu$. The population of λ individuals is generated by selecting past individuals through tournament selection, crossover and random mutation, where 1 gene is mutated, selected uniformly at random. When mutating the opponents plan in RHCP, M is set to 5. The default MCTS agent, included in the GVGAI framework, is used for the experiments. This is a closed loop version [17] of the algorithm, using a standard UCB1 [13] function for its tree policy. Uniform random roll-outs, limited to a depth of 15, are used for the default policy. Finally, the random agent is also taken verbatim from the provided agents and it simply chooses one of the available moves in the current state uniformly at random.

RHEA, MCTS and RHCP substituted the time-based budget from the competition (40ms of real-time per decision) for a number of usages of the forward model's function that rolls the state forward. This ensures that all experiments are consistent independently from the machine used. As in [6], 900 FM calls were provided as the decision-making budget for each game tick. For each game, all 5 levels were evaluated with 2 different orders, each agent playing both sides, providing 10 different scenarios. Each scenario is then played 10 times, resulting in 100 trials per agent pairing on each game, and a total of 1000 matches per pair of controllers.[2]

Results

This experiment compares our proposed RHCP algorithm against a suite of control algorithm as well as investigating if co-evolution helps predict the opponent's next move. The overall win/loss results for all ten games are shown in Table 2 and the prediction accuracy against each opposing algorithm is in Table 3.

Comparison of Agent vs Agent performance

RHCP vs RHEA

Comparing RHCP to RHEA is perhaps most interesting as RHCP operates similarly to RHEA, with the addition of co-evolving an opponent's plan at the expense of some computational time. Of the 1000 games played, RHCP won 360 compared to 309 games won by RHEA. These 51 games, or 5.1% increase is a significant ($p < 0.05$) improvement. A 2-sample Z test was carried out for significance testing with a Z-Score of 2.417 and a p-value of 0.015.

RHCP/RHEA vs MCTS

MCTS is the most dominant two-player GVGAI approach and it is clear that whilst RHCP performs better than RHEA, the performance of both is considerably worse than MCTS. Neither RHCP nor RHEA is significantly better than one another when playing against MCTS. It does seem that RHCP is more competitive than RHEA against MCTS (winning more and losing less), although further testing would be required to confirm this, especially due to the stochastic nature of evolutionary algorithms.

RHCP/RHEA/MCTS vs Random

Understandably RHCP, RHEA, and MCTS all performed very well against a random agent. It is interesting to note that even against a random agent, no other agent was able to achieve a better than 50/50 win ratio in game 4 - *Gotcha*. We believe this because not only is *Gotcha* asymmetric but also unbalanced - different 'roles' provide different levels of challenge.

Prediction Accuracy

The opponent's predicted next move based on the first move of the RHCP evolved opponent's plan and the opponent's actual move were recorded for each tick in games involving RHCP. Table 3 shows the average accuracy of these predictions for each game played against each opponent.

Discussion

RHCP outperforms RHEA using a random opponent model and also closes the gap between RHEA and MCTS - only slightly. RHCP performed as well as or better than RHEA in every game, in terms of win % indicating that spending time planning for the opponent's is worthwhile, even in environments such as GVGAI where there is a very limited computational budget. It is interesting to note that the two games where RHCP really performed well - game 2, *Capture the Flag* and game 9, *Tron* - are particularly adversarial; the actions the opponent takes have a big impact on the performance of the agent. Additionally, RHCP and RHEA performed similarly, within 4 wins of each other, for all games except game 5, *Klax*. Whilst this win ratio improvement is not significant, $p = 0.061$, this is another example of RHCP performing better in games which are highly interactive and are reliant on your opponent's moves, exactly the sorts of games RHCP can give a competitive advantage.

However, MCTS is still the best performing agent. Rolling Horizon agents can be sensitive to both hyper-parameter choices and features of the game. For this reason, options such as games with higher branching factors or that need longer plans may well favour RHEA. Further testing on a wider range of games and hyper-parameter options would be needed to check this hypothesis.

The prediction accuracy was not significantly different amongst the various opponents. Given that one agent is a Random agent, this essentially means that across all games predicting the next move based on co-evolution is no better than random. That said game 4, *Gotcha* is a game where RHCP was able to predict the next move

[2] Tests were run in the Apocrita High Performance Compute cluster[12]

Table 1. Games set from the GVGAI Two-Player Training Set 1. The key game attributes are Cooperative (Co) or Competitive (Cp), Symmetric (Sym) or Asymmetric (Asym), and Stochastic (S) or Deterministic (D). The descriptions are summaries of the descriptions found at www.gvgai.net.

ID	Game	Key Attributes	Description
0	Akka Arrh	Co, Sym, S	Two players defend a locked spaceship from aliens, while trying to open and enter in it. Both players win when they enter the spaceship but lose if one is hit by the aliens.
1	Asteroids	Cp, Sym, S	Players shoot asteroids, which break into smaller asteroids, for points. The players can also shoot each other. Last player standing wins, or the one with more points at the end.
2	Capture the Flag	Cp, Sym, D	The level is divided into two areas, each with a flag. Players must capture the opponent's flag and bring it to their area. Capturing gives points, the player with most points wins.
3	Cops N Robbers	Cp, Asym, D	One player is the robber and the other is a cop. The robber wins if all gems are collected and the cop wins if he catches the robber.
4	Gotcha	Cp, Asym, S	One player has to chase the other. There are safe places where the chased one can hide. If the chaser catches the other player, they win but loses if time is over before that.
5	Klax	Cp, Sym, S	Coloured objects fall from the sky. Players catch them for points (own colour is worth more points). Higher score at the end decides the winner.
6	Samaritan	Cp, Asym, D	One player tries to cross a portal to another world, while the other tries to avoid so. The first player wins the game by reaching it on time.
7	Sokoban	Co, Sym, D	Both players must push all boxes into determined locations. The game ends when all boxes are correctly placed.
8	Steeplechase	Cp, Sym, D	Racing game. Players win by reaching the end. There are many obstacles and a hidden gem is worth many score points.
9	Tron	Cp, Sym, D	A version of the classic game with the same name. Players race in a wall-encircled arena creating walls as they move. The first player that collides with a wall loses.

Table 2. Results from all games played across the 6 pairings of agents. For each pair, first (second) column shows number of victories for the first (second, respectively) agent. The maximum possible wins for each pairing and game is 100 (ties possible).

Game ID	RHCP vs RHEA	RHCP vs MCTS	RHEA vs MCTS	RHCP vs Rand.	RHEA vs Rand.	MCTS vs Rand.
0	0 - 0	1 - 1	3 - 3	0 - 0	0 - 0	1 - 1
1	49 - 45	9 - 85	7 - 87	85 - 9	92 - 6	100 - 0
2	56 - 45	6 - 97	3 - 98	88 - 12	90 - 11	100 - 0
3	39 - 31	3 - 66	4 - 70	59 - 8	57 - 13	69 - 1
4	50 - 50	50 - 50	50 - 50	50 - 50	50 - 50	50 - 50
5	54 - 48	35 - 70	23 - 79	100 - 0	97 - 4	100 - 0
6	50 - 50	46 - 54	50 - 50	56 - 38	53 - 44	64 - 34
7	0 - 0	0 - 0	0 - 0	0 - 0	0 - 0	0 - 0
8	1 - 1	1 - 3	4 - 3	2 - 1	3 - 4	12 - 2
9	61 - 39	6 - 94	10 - 90	98 - 2	98 - 2	100 - 0
Total	**360 - 309**	**157 - 520**	**154 - 530**	**538 - 120**	**540 - 134**	**596 - 88**

with a high average accuracy against the RHEA agent. The prediction accuracy was 0.418, compared to 0.199 against random, a significant improvement (Z Score = 3.36, $p < 0.05$). Even though the average move prediction accuracy against RHEA/MCTS is not high, co-evolution is clearly making a positive difference to decision making, evidenced by RHCP agent performing better than RHEA. One possible reason for the poor prediction accuracy is the stochastic nature of the evolutionary algorithm coupled with the small number of evolution iterations, given such limited forward model calls: RHCP performs approximately 6 generations per tick. It is possible that the prediction plan may find a strong move sequence which is still valuable, but not the exact move sequence found by the opponent.

Conclusion

In this paper, we discuss Rolling Horizon Co-evolutionary Planning (RHCP) for General Video Game Playing, which evolves plans of actions for itself and the opponent. We have experimented and compared this algorithm to three others, RHEA, MCTS and Random in a set of 10 two-player GVGAI games. We have found that spending computation time considering the potential opponent's moves shows improvement compared to the same algorithm without this feature (RHEA). However, this still performed worse than MCTS. Surprisingly, despite RHCP outperforming RHEA, it was not able to predict the next move of 'rational' agents (RHEA/MCTS) with a greater accuracy than against Random. Perhaps the opponent plan evolution

Table 3. Prediction accuracy (Acc $\in [0, 1]$) for the first move in the RHCP opponent plan compared to the actual move taken, together with the Standard Deviation, in parenthesis, across all trials per game against all opposing agents. Highest accuracy per game is show in bold.

Game	RHEA	MCTS	Random
0	**0.185** (0.016)	0.165 (0.016)	0.166 (0.02)
1	0.169 (0.107)	0.160 (0.167)	**0.173** (0.15)
2	**0.223** (0.01)	0.201 (0.010)	0.2 (0.01)
3	**0.2** (0.032)	0.176 (0.031)	0.18 (0.033)
4	**0.418** (0.013)	0.202 (0.041)	0.199 (0.025)
5	0.375 (0.013)	0.332 (0.013)	**0.418** (0.013)
6	**0.201** (0.034)	0.180 (0.044)	0.181 (0.026)
7	**0.223** (0.009)	0.199 (0.009)	0.199 (0.009)
8	**0.186** (0.008)	0.166 (0.009)	0.168 (0.009)
9	0.245 (0.06)	**0.227** (0.045)	0.197 (0.102)
Avg	**0.223** (0.032)	0.201 (0.038)	0.208 (0.04)

is finding strong sequences over 15 moves but, due to the stochastic nature of RHCP, it often disagrees on one move in particular. This work should be seen as a first step towards applying RHCP effectively. Thus, further investigation is required to fully understand the positive effects of co-evolution, though it's reasonable to think that predicting the opponent's move more accurately would improve the performance of RHCP. Future work should investigate how existing RHEA modifications impact RHCP - including probabilistic approaches to further improve predictions of actions - what are the best hyper-parameters and how differently it would perform in more games. Also, submitting RHCP to the GVGAI competition would allow assessing its performance against other approaches.

Acknowledgments

This work was supported by grant EP/L015846/1 for the Centre for Doctoral Training in Intelligent Games and Game Intelligence (IGGI - http://www.iggi.org.uk/) from the UK Engineering and Physical Sciences Research Council (EPSRC). This research utilised Queen Mary's Apocrita HPC facility, supported by QMUL Research-IT.

REFERENCES

[1] Marc G Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling, 'The arcade learning environment: An evaluation platform for general agents.', *J. Artif. Intell. Res.(JAIR)*, **47**, 253–279, (2013).

[2] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba, 'Openai gym', *arXiv preprint arXiv:1606.01540*, (2016).

[3] Cameron B Browne, Edward Powley, Daniel Whitehouse, Simon M Lucas, Peter I Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton, 'A survey of monte carlo tree search methods', *IEEE Transactions on Computational Intelligence and AI in games*, **4**(1), 1–43, (2012).

[4] Raluca D Gaina, Adrien Couëtoux, Dennis JNJ Soemers, Mark HM Winands, Tom Vodopivec, Florian Kirchgeßner, Jialin Liu, Simon M Lucas, and Diego Perez-Liebana, 'The 2016 two-player gvgai competition', *IEEE Transactions on Computational Intelligence and AI in Games*, (2017).

[5] Raluca D. Gaina, Simon M. Lucas, and Diego Perez-Liebana, 'Population seeding techniques for Rolling Horizon Evolution in General Video Game Playing', *2017 IEEE Congress on Evolutionary Computation, CEC 2017 - Proceedings*, 1956–1963, (2017).

[6] Raluca D Gaina, Simon M Lucas, and Diego Pérez-Liébana, 'Population seeding techniques for rolling horizon evolution in general video game playing', in *Evolutionary Computation (CEC), 2017 IEEE Congress on*, pp. 1956–1963. IEEE, (2017).

[7] Raluca D. Gaina, Simon M. Lucas, and Diego Perez-Liebana, 'Rolling horizon evolution enhancements in general video game playing', *2017 IEEE Conference on Computational Intelligence and Games, CIG 2017*, 88–95, (2017).

[8] Michael Genesereth, Nathaniel Love, and Barney Pell, 'General game playing: Overview of the aaai competition', *AI magazine*, **26**(2), 62, (2005).

[9] Jose Manuel Gonzalez-Castro and Diego Perez-Liebana, 'Opponent models comparison for 2 players in gvgai competitions', in *Computer Science and Electronic Engineering Conference (CEEC)*, (2017).

[10] Ahmed Khalifa, Michael C Green, Diego Pérez-Liébana, and Julian Togelius, 'General Video Game Rule Generation', in *2017 IEEE Conference on Computational Intelligence and Games (CIG)*. IEEE, (2017).

[11] Ahmed Khalifa, Diego Perez-Liebana, Simon M Lucas, and Julian Togelius, 'General video game level generation', in *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*. ACM, (2016).

[12] Thomas King, Simon Butcher, and Lukasz Zalewski, 'Apocrita - high performance computing cluster for queen mary university of london', (Mar 2017).

[13] Levente Kocsis and Csaba Szepesvári, 'Bandit based monte-carlo planning', in *European conference on machine learning*, pp. 282–293. Springer, (2006).

[14] Jialin Liu, Diego Pérez-Liébana, and Simon M. Lucas, 'Rolling horizon coevolutionary planning for two-player video games', *2016 8th Computer Science and Electronic Engineering Conference, CEEC 2016 - Conference Proceedings*, 174–179, (2017).

[15] Santiago Ontanón, Gabriel Synnaeve, Alberto Uriarte, Florian Richoux, David Churchill, and Mike Preuss, 'A survey of real-time strategy game ai research and competition in starcraft', *IEEE Transactions on Computational Intelligence and AI in games*, **5**(4), 293–311, (2013).

[16] Diego Perez, Spyridon Samothrakis, Simon Lucas, and Philipp Rohlfshagen, 'Rolling horizon evolution versus tree search for navigation in single-player real-time games', *Proceeding of the fifteenth annual conference on Genetic and evolutionary computation conference - GECCO '13*, 351, (2013).

[17] Diego Perez Liebana, Jens Dieskau, Martin Hunermund, Sanaz Mostaghim, and Simon Lucas, 'Open loop search for general video game playing', in *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*, pp. 337–344. ACM, (2015).

[18] Diego Perez-Liebana, Jialin Liu, Ahmed Khalifa, Raluca D. Gaina, Julian Togelius, and Simon M. Lucas, 'General Video Game AI: a Multi-Track Framework for Evaluating Agents, Games and Content Generation Algorithms', (2018).

[19] Diego Perez-Liebana, Spyridon Samothrakis, Julian Togelius, Simon M Lucas, and Tom Schaul, 'General video game ai: Competition, challenges and opportunities', in *Thirtieth AAAI Conference on Artificial Intelligence*, pp. 4335–4337, (2016).

[20] Diego Perez-Liebana, Spyridon Samothrakis, Julian Togelius, Tom Schaul, Simon M Lucas, Adrien Couëtoux, Jerry Lee, Chong-U Lim, and Tommy Thompson, 'The 2014 general video game playing competition', *IEEE Transactions on Computational Intelligence and AI in Games*, **8**(3), 229–243, (2016).

[21] Christopher D. Rosin and Richard K. Belew, 'New methods for competitive coevolution', *Evol. Comput.*, **5**(1), 1–29, (March 1997).

[22] Tom Schaul, 'A video game description language for model-based or interactive learning', in *Computational Intelligence in Games (CIG), 2013 IEEE Conference on*, pp. 1–8. IEEE, (2013).

[23] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al., 'Mastering the game of go with deep neural networks and tree search', *nature*, **529**(7587), 484–489, (2016).

[24] Julian Togelius, Noor Shaker, Sergey Karakovskiy, and Georgios N Yannakakis, 'The mario ai championship 2009-2012', *AI Magazine*, **34**(3), 89–92, (2013).

[25] Ruben Rodriguez Torrado, Philip Bontrager, Julian Togelius, Jialin Liu, and Diego Perez-Liebana, 'Deep reinforcement learning for general video game ai', in *IEEE Conference on Computational Intelligence and Games (CIG)*. IEEE, (2018).