Rolling Horizon Evolution Enhancements in General Video Game Playing

Raluca D. Gaina
University of Essex
Colchester, UK
Email: rdgain@essex.ac.uk

Simon M. Lucas
University of Essex
Colchester, UK
Email: sml@essex.ac.uk

Diego Perez-Liebana
University of Essex
Colchester, UK
Email: dperez@essex.ac.uk

***Abstract*—Game AI literature has looked at applying various enhancements to Rolling Horizon Evolutionary methods or creating hybrids with popular tree search methods for an improved performance. However, these techniques have not been analyzed in depth in a general setting under the same conditions and restrictions. This paper proposes a fair juxtaposition of four enhancements applied to different parts of the evolutionary process: bandit-based mutation, a statistical tree for action selection, a shift buffer for population management and additional Monte Carlo simulations at the end of an individual's evaluation. These methods are studied individually, as well as their hybrids, on a representative subset of 20 games of the General Video Game AI Framework and compared to the vanilla version of the Rolling Horizon Evolutionary Algorithm, in addition to the dominating Monte Carlo Tree Search. The results show that some of the enhancements are able to produce impressive results, while others fall short. Interesting hybrids also emerge, encouraging further research into this problem.**

I. Introduction

Academic interest for Artificial General Intelligence (AGI) has spread across Game AI research during the last years. With the objective of creating AI that can play multiple games, rather than specifically tackling single problems one at a time, researchers are trying to push the boundaries of AGI by bringing new methods and testbeds. Examples are the Arcade Learning Environment (ALE), where Deep Reinforcement Learning techniques have been able to reach human level of play [1], or the General Video Game AI (GVGAI[1]) Framework and Competition [2], [3]. GVGAI proposes a benchmark for planning, learning and procedural content generation that has attracted multiple authors within the last few years.

Rolling Horizon methods for planning have raised during this time as an alternative to tree search for real-time control in games, particularly in the domain of GVGAI. Although tree-based search methods have been, in most cases, proclaimed winners of different GVGAI tracks [2], recent research in Rolling Horizon Evolutionary Algorithms (RHEA) has closed the gap with the former ones [4], [5].

This paper aims to explore four enhancements to the vanilla RHEA. Some of the enhancements presented here have been seen in the literature before, either in a General Video Game Playing (GVGP) setting, or in some other domains. However, they have been previously employed under different conditions, heuristics and set of games, and sometimes combined with other techniques. It is therefore very hard to deduct (if not impossible) which ones of these approaches work well in isolation, which ones do not produce improvements in the vanilla form of the algorithms, if decoupled from some heuristics properly used, and which ones could work better if put together in combination.

The objective of this paper is to formalize and provide a fair analysis on these enhancements. They are all tested in isolation, but also combinations between them are drawn in order to identify potentially good synergies. Furthermore, they are evaluated under the same testing circumstances, provided with a common heuristic to evaluate states visited during search, and in 20 GVGAI games carefully selected to serve as a good representative set of the whole GVGAI corpus.

The rest of this document is structured as follows. Section II gives an overview of recent works in this domain. An explanation of the framework and the techniques behind the agents employed in this study is given in Section III. Later, Section IV looks at the enhancements proposed in this paper for RHEA, while Section V describes the experimental approach. Section VI discusses the results obtained and, finally, Section VII concludes the paper and outlines future work.

II. Literature Review

Rolling Horizon Evolutionary Algorithms (RHEA) were compared by Perez et al. in [6] with Monte Carlo Tree Search (MCTS), on a specific real-time game, the Physical Travelling Salesman Problem (PTSP). They used simple macro actions to more easily explore the large game space and analyze the performance of the RHEA with three different mutation rates, while fixing the population size and individual lengths. Their results show that RHEA is a promising competitor to MCTS. Additionally, Samothrakis et al. have compared algorithms similar to these two methods on three continuous games in [7] with satisfactory results in favor of evolution.

Recent Game AI literature looked at combining evolution and tree search in interesting ways in order to make use of the benefits of both methods. Lucas et al. [8] applied an evolutionary process to guide the simulation step of MCTS and improve upon the random default policy. They show results on both the Mountain Car problem and a simple

[1] www.gvgai.net

version of Space Invaders, seeing a significant increase in performance. A similar technique was later employed in the General Video Game AI Framework (GVGAI) by Perez et al. [9], together with a knowledge base aimed at maximizing the information gain from the limited thinking time. The algorithm's performance is again observed to increase due to the combination of the two techniques.

However, in this paper we are looking at the effects of the reverse process, the integration of other systems into the evolutionary algorithm instead. Gaina et al. [5] have looked at the possibility of using MCTS in the initialization step of RHEA. In this setting, MCTS would take half of the budget to recommend a solution, which would then be used by the main algorithm as the starting point for evolution. This method produced good results, significantly out-performing the vanilla version of RHEA and getting closer to the dominant performance of MCTS.

Additionally, Perez et al. [10] keep a statistical tree alongside the evolutionary process, in order to record statistics about the actions while evaluating individuals and select the action with the highest value averaged during the evolution. Therefore, they make use of intermediate states and not only look at the final population obtained. This method is most effective in noisy environments as an alternative to resampling, which would be more expensive. Furthermore, they keep the tree from one game step to the next, by using the child selected at the end of the evolution as the new root of the tree in the following step. Their promising results motivated the use of both of these methods in this study, by combining the stats tree with a shift buffer for the same effect. However, it is worth noting that the authors add a pheromone-based heuristic to their algorithms which may impact their findings.

A compelling and novel addition to evolutionary algorithms is that of multi-armed bandits applied as a mutation operator to better balance between exploration and exploitation. There is extensive literature on the multi-armed bandit problem [11] and various solutions to it. One possibility is using an Upper Confidence Bound (UCB) method. Powley et al. [12] look at using UCB in Monte Carlo Tree Search as both the tree policy and the simulation policy. When tested on three different problems, two card games ("Dou Di Zhou" and "Hearts") and a board game ("Lord of the Rings: The Confrontation"), its performance is shown to consistently be at a high level.

The RHEA variant presented in this paper employs a bandit-based mutation system as described in [13], [14]. Liu et al. compare this mutation method with the Random Mutation Hill Climber (RMHC) on two simple problems and their results suggest that bandit-based mutation is especially effective in cases where individual evaluation is expensive, therefore applicable to the problems described in the present paper. This work will also expand from the RMHC to a larger population of individuals, in order to assess how this type of mutation is affected by an increase in core parameter values.

Horn et al. [15] look at two different MCTS-RHEA hybrids. In the first method (*EAroll*), Monte Carlo simulations are used at the end of the evaluation of one RHEA individual with a limited depth, the resulting value being averaged with the genome evaluation to determine its fitness. The second variant (*EAaltActions*) uses both RHEA and MCTS to individually search for distinct solutions, the two final recommendations being evaluated and the best one chosen for execution. They analyse the performance of both algorithms on 20 games of the GVGAI corpus (but a different 20 than in our work) and *EAroll* appears to be significantly better than vanilla RHEA and dominating the games used in their experiments.

III. Background

A. The General Video Game AI Framework

The General Video Game AI Framework (GVGAI) was used as the testbed for the experiments reported in this paper. It comprises of a large number of real-time 2D grid games (currently 100 single player and expanding to continuous physics games with a new set of 10 games). Therefore, it is a great environment for observing the performance of intelligent agents on multiple highly-varied problems.

The types of games range from classic arcade (Aliens, a version of Space Invaders), to puzzle (Sokoban), shooters and many more. They differ in the way players are able to interact with the environment (they may have different actions available in certain games, such as movement and special actions), the scoring system, the objects part of a game (NPCs, resources etc.) or the end game conditions.

The information received by AI agents is limited to the current game state, leaving it up to them to figure out the rules. However, they also have access to a Forward Model (FM), which can be used to look into possible future states and access more knowledge about what may happen. As some of the games are stochastic, the FM is not guaranteed to provide a perfect representation of the next state.

Once all controllers have played on a given game, they are sorted by average of victories first, followed by score and average time they took to finish a game. According to their position, the agents receive 25, 18, 15, 12, 10, 8, 6, 4, 2 and 1 points, from the first to the tenth ranked player, with the rest receiving 0 points, as in a Formula-1 (F1) system. When compared across different games, the winner is determined by summing the points obtained in all of the games.

B. Evolutionary Algorithms

Evolutionary Algorithms (EA) are a large family of algorithms inspired from biological sciences. They encode solutions to problems as individuals, part of a population which evolves over several generations, until a good enough solution, as defined by the specific problem, is found or an execution limit is reached. In the setting used in this study, individuals are simply sequences of actions to be executed in the game. The model adopted in this paper was that of Rolling Horizon Evolutionary Algorithms (RHEA) [6], which begins at each game tick with a new set of action plans and evolves them through different techniques.

The simplest method for population initialization is random, although others were analyzed by Gaina et al. [5] under the

same circumstances used in this study and in the same set of games. At the end of the execution budget, the agent chooses to play the first action from the best plan evolved. One plan of actions is evaluated by making use of the FM offered in GVGAI and simulating ahead through the actions, one at a time. The game state reached at the end of the sequence is evaluated with a heuristic function, this value becoming the fitness of the individual.

C. Bandits

The multi-armed bandit problem [16] is a classic problem, in which a gambler having access to multiple machines needs to make a decision as to which machine's lever they should pull. Each machine produces a random reward from a specific probability distribution. The goal of the gambler is to maximize the sum of rewards obtained through subsequent plays. Therefore they need to balance their exploration and exploitation, in order to learn the different distributions, while getting the maximum benefits from their plays.

One of the solutions to the problem and the method employed in this paper is using the UCB (Upper Confidence Bound) equation (Equation 1). The first term ($Q(s,a)$) attempts to maximize the value of the play (exploitation). The second term favors levers which were pulled the least number of times (exploration), $N(s,a)$ indicating the number of times lever a was pulled and $N(s)$ the total number of plays. The constant C is that which balances between the two terms and it may be adjusted to fit specific problems.

$$a^* = \underset{a \in A(s)}{\arg\max} \left\{ Q(s,a) + C\sqrt{\frac{\ln N(s)}{N(s,a)}} \right\} \quad (1)$$

In GVGAI, levers are represented by actions, therefore, from one state, the UCB equation would ensure that good actions are chosen, while exploring those not chosen as often to analyse their effect and build up the knowledge base.

D. Monte Carlo Tree Search

The dominant techniques in GVGAI are mainly based on Monte Carlo Tree Search (MCTS) [17]. The sample version provided with the framework that many competition entries are based on is an MCTS variant using four steps at each iteration and UCB as the tree policy (Equation 1).

At each game step, the algorithm begins by creating a root node to its search tree. Then each iteration consists of four steps: selection, expansion, simulation and back-propagation. MCTS first selects a non-terminal and not fully expanded node using the tree policy. This node is then expanded by adding a new child (by choosing an action to take, which would lead to a new game state). From the newly added node, actions are randomly selected to play through the game, using the FM to simulate ahead. Finally, the state reached after the simulation step is evaluated using a heuristic and its value is used to update all of the nodes that have been visited during this iteration, up to the root of the tree.

These iterations are repeated until an execution budget limit is reached. The algorithm returns the child of the root node that is considered the best (e.g. highest value or most visited).

As some of the games in the GVGAI Framework are stochastic, an open loop approach is preferred, which only stores statistics in the nodes of the tree and not the actual game states. This is the variant inspiring one of the enhancements presented in this paper and also that to which the RHEA algorithms are compared to in Section VI-C.

IV. Rolling Horizon Evolution Enhancements

The baseline algorithm is *Vanilla RHEA*. The population initialization is kept pseudo-random (each individual receiving random actions for each gene, in the range $0-(N-1)$, where N is the number of legal actions in the current game state (therefore each gene corresponds to one in-game action).

Breeding occurs $P - E$ times in one generation, where E represents elitism (the chosen method for promoting the best individuals, unchanged, to the next generation. $E = 1$ for all cases) and P population size. Each new individual in a subsequent generation is the product of uniform crossover between individuals from the previous generation, selected through tournament (size 2), and mutation (random).

The heuristic used to evaluate game states and determine individual fitness simply returns the game score, dynamically normalized between 0 and 1, or a large reward for winning (and a large penalty for losing, respectively). The process of evaluating an individual is as described in Section III.

In the rest of this paper, the term "configurations" will refer to population size (P) and individual length (L) values and the term "variants" will refer to RHEA algorithms with enhancements added to the vanilla version. If more than one enhancement is used, the term "hybrid" may be used instead.

A. Bandit-based mutation

The first of the enhancements analyzed in this study is using a bandit system for individual mutation, employing the UCB technique with the constant $C = \sqrt{2}$. This algorithm will be referred to as *EA-bandit*.

In the RHEA variants with bandit mutation (identified in the results discussion by having the term "bandit" in their name), two levels of bandit systems are used.

The first system is at individual level, used to select which gene to mutate. In the exploration term from the UCB equation (Equation 1), $N(s,a)$ is the number of times gene a was mutated and $N(s)$ is the total number of mutations. The exploitation term is determined by $max(\Delta R)$, the maximum difference in rewards observed when mutating gene a. The differences in rewards are updated after each mutation by evaluating the new individual obtained. If the new ΔR is negative (thus there was no improvement in the value of the individual), the mutation is reverted. Therefore, the individuals will never get worse with this mutation operator.

The second system is at gene level (therefore L bandits, one for each gene). The information from all of the individuals in the population is stored in the same set of L bandits as they all

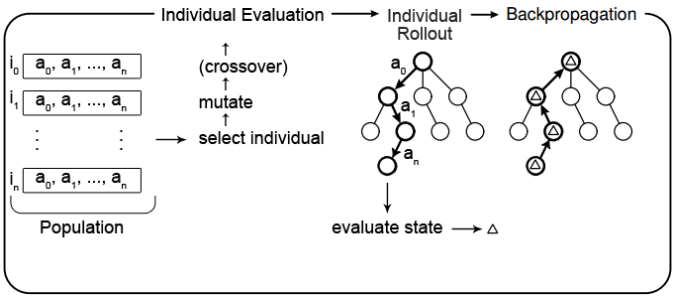

Figure 1: RHEA statistical tree steps.

aim to find the same optimal action plan. Therefore, a number $P - E$ values are used for updating the bandit information each generation. In this case, the exploration term is made up of the number of times gene X was changed ($N(s)$) and the number of times gene X received value a ($N(s, a)$). The exploration term is the ΔR corresponding to the value a.

When combined with the statistical tree, this enhancement remains unchanged. However, when combined with the shift buffer, the gene-level bandits are shifted along with the population in the same manner. Additionally, all ΔR values are discounted by a factor $\gamma = 0.99$.

B. Statistical tree

This enhancement (*EA-tree*) keeps a statistical tree alongside the population evolved by RHEA, similar to the work in [10]. Every time an individual is evaluated, its actions are used to traverse the tree (new nodes being added if new actions are encountered). The fitness value is used to update the statistics stored in each node that has been visited during the individual evaluation. This process is depicted in Figure 1. $(P - E) \times L$ nodes are updated at each generation.

This stats tree comes into play when choosing which move to make at the end of the evolution. The vanilla version of the algorithm returns the first action of the best individual found. With this enhancement, the action returned is the child with the highest UCB value (Equation 1), aiming for a better balance between exploitation and exploration of the search space.

The RHEA variants with a stats tree are identified in the results discussion by having the term "tree" in their name.

When combining this enhancement with bandit-based mutation, the process remains unchanged. However, combining it with the shift buffer results in the tree being trimmed and carried forward instead, the best child selected becoming the root of the tree in the next game step, while its siblings are discarded, in a similar manner as in [10]. If the tree is kept, the values stored in its nodes are discounted by a factor $\gamma = 0.99$.

C. Shift buffer

The shift buffer enhancement (*EA-shift*) is a simple technique which aims at maximising the information gain in the limited thinking time received by the algorithms, by keeping information from one game tick to the next, instead of starting from scratch, as is the case in the vanilla version.

Therefore, this method shifts the population of individuals obtained at the end of one game tick to the left and adds a new random action at the end of each individual in the population. If the number of legal actions is reduced from one game step to the next, then the genes surpassing the new maximum number are replaced by new random legal actions.

The RHEA variants with a shift buffer are identified in the results discussion by having the term "shift" in their name.

D. Rollouts

This enhancement (*EA-roll*) is inspired by Monte Carlo simulations and the work described in [15]. Therefore, when evaluating an individual, instead of stopping at the last action in the sequence and valuing the state reached at that point, the process continues with random selection of actions and game simulations using the FM model (discounted from the total budget). This final game state reached is evaluated instead, with the same heuristic function, and its value becomes the fitness of the individual.

The motivation behind the use of additional rollouts lies in the fact that the algorithm receives a further look ahead, without being restricted to only a specific set of actions (as it is the case when the L parameter value is increased directly).

The length of the rollouts used in this study is $L/2$ in all cases. This process may be repeated a number of times $R = \{1, 5, 10\}$ and the values obtained averaged over all repetitions.

The RHEA variants with rollouts are identified in the results discussion by having the term "roll" in their name.

V. Experimental Setup

Several variations of the vanilla RHEA algorithm were analyzed on a set of 20 games (see Section V-A), playing 20 times on all 5 levels of each game (therefore 100 runs per game per algorithm). Additionally, 4 different core parameter configurations (P-L = {1-6, 2-8, 5-10, 10-14}) were used for all algorithms, in order to observe the effect of the enhancements across a range of parameter values, comparable with the results presented in [4], [5].

The budget given to each algorithm was restricted to 900 FM calls (the average obtained by vanilla RHEA in the current GVGAI corpus), so as to eliminate bias from variations in the machine used to run the experiments. The maximum configuration tested was 10-14 due to the fact that if it were larger, by adding rollouts, the limited budget would not allow for even one full population to be evaluated in one game tick.

There are two main parts to the experiments run for this study, the second of which includes a comparison with MCTS. The results presented in Section VI correspond to this setup.

The first part of the experiments explored the first three enhancements described in Section IV (bandit-based mutation, stats tree and shift buffer) in isolation, as well as combinations of them, resulting in 8 variants. The best variants in all configurations (4 in total) were kept for the next part.

The second part of experiments looked at the last enhancement (rollouts, see Section IV-D), added to the 4 variants promoted previously. Three different values for rollout repetitions

Idx	Name	Type	Idx	Name	Type
0	Aliens	S	4	Bait	D
13	Butterflies	S	15	Camel Race	D
18	Chase	D	22	Chopper	S
25	Crossfire	S	29	Dig Dug	S
36	Escape	D	46	Hungry Birds	D
49	Infection	S	50	Intersection	S
58	Lemmings	D	60	Missile Command	D
61	Modality	D	67	Plaque Attack	D
75	Roguelike	S	77	Sea Quest	S
84	Survive Zombies	S	91	Wait for Breakfast	D

Table I: Names, indexes and types of the 20 games from the subset selected. Legend: S = Stochastic, D = Deterministic.

were considered: $R = \{1, 5, 10\}$. This resulted in 8 algorithms (with and without rollouts) analyzed in this section for each rollout length, therefore 24 per configuration.

Finally, the algorithms were compared with MCTS in order to validate their quality on a larger scale in GVGAI.

A. Game set

The game set employed in this study is a selection of games from the GVGAI corpus, based on two different studies which classified a large number of games according to the performance of various algorithms. M. Nelson looked at 62 games and ranked them in relation to the performance of the vanilla MCTS algorithm [18]. Bontrager et al. analyzed 49 games and used a clustering method to group them relatively to the their perceived difficulty, as dictated by the results of several competition entries [19].

The 20 games were uniformly sampled from both works. Moreover, 10 of the games in the set are deterministic and 10 are stochastic (see Table I). The resulting game set is highly diverse and offers a good range of problems.

VI. Results

The results presented in this section are based on both rankings following the Formula-1 point system (see Section III-A) and a significance comparison in win rate or scores, using a Mann-Whitney non-parametric U test, with p-value $= 0.05$.

A. Bandit, tree, shift

Overall, in the first part of experiments, the shift buffer appears to offer the biggest improvement in performance, while the bandit-based mutation is in many cases significantly worse than all other algorithms. If all variants across all configurations were to be compared and ranked according to F1 points, *EA-shift* (5-10) would be in first place, with 213 points and 40.05% average win rate, while *EA-bandit* (1-6) would be last, with 0 points and 29.65% win rate. The specific results for configuration 5-10 are depicted in Table II.

The effect of increasing the population size and individual length is noticed in most variants. Although the win rate sees an overall increase proportional to parameter values, the algorithm ranking does not remain consistent.

Figure 2 presents the significant wins of all variants in all configurations, counting for each pair in how many games the row algorithm was significantly better than the column one; the darker the color, the higher the game count. A dark row would therefore signify an algorithm better than the others in most games, while a dark column would mean the algorithm performed worse. It is worth observing how bandit hybrids feature dark columns in most configurations, as well as how *EA-shift* and *EA-tree-shift* rows stand out as the best.

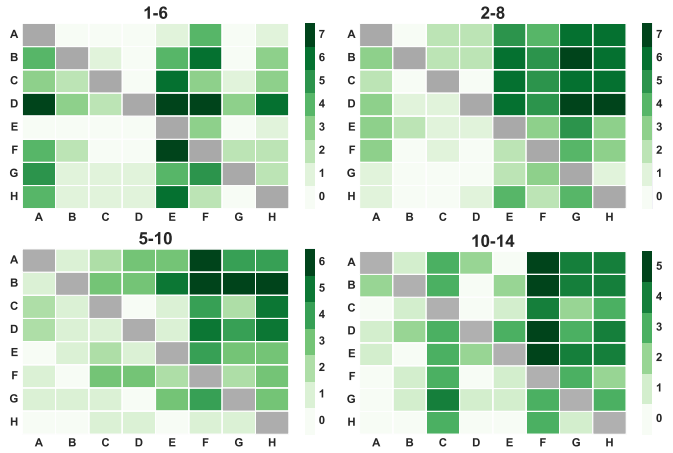

Figure 2: Win percentage for all configurations. The color bar denotes in how many unique games row was significantly better than column. Legend: A = Vanilla, B = EA-shift, C = EA-tree, D = EA-tree-shift, E = EA-bandit, F = EA-bandit-shift, G = EA-bandit-tree, H = EA-bandit-tree-shift

An interesting game to look at in more detail is game 60 (*Missile Command*), where no significance can be observed in low configurations, but in higher ones *EA-shift* is significantly better than vanilla in win rates and all shift hybrids are better than vanilla in scores; *EA-bandit* is significantly worse than both shift and tree hybrids. In game 36 (*Escape*), all variants are significantly better than vanilla in both win rates and scores, except for tree hybrids, in low configurations, while no significance is observed at the opposite end of the spectrum.

In most games, the shift enhancement is significantly better across configurations, *EA-shift* (2-8 and higher) being able to match and surpass the performance of the best *Vanilla RHEA* (10-14). This is a critical finding of this study: the simple shift buffer enhancement, which requires little extra computation time, allows for much better performance without needing to increase core parameter values.

The best 4 algorithms carried forward to the second part of experiments are *EA-shift*, *EA-tree-shift*, *EA-tree* and *Vanilla*.

1) EA-bandit: The *EA-bandit* algorithm is one of the worst variants tested in this study. In all configurations, it performed worse than *Vanilla* and, in the smallest configuration (1-6), it was out-performed by all of the bandit hybrids as well. However, in higher configurations it increases its average win rate significantly, from 29.65% to 38.50% and even outperforms *EA-tree* in the largest configuration (10-14).

In game 67 (*Plaque Attack*), *EA-bandit* attains a significantly better win rate than most algorithms, increasing from

#	Algorithm	Points	Avg. Wins	G-0	G-4	G-13	G-15	G-18	G-22	G-25	G-29	G-36	G-46	G-49	G-50	G-58	G-60	G-61	G-67	G-75	G-77	G-84	G-91
1	**EA-shift**	373	40.05 (2.50)	18	15	18	15	**25**	**25**	12	**25**	18	12	18	15	**25**	**25**	6	**25**	**25**	8	**25**	18
2	**EA-tree-shift**	293	37.20 (2.48)	**25**	8	15	18	18	8	18	10	10	**25**	10	8	6	18	12	18	18	**25**	15	8
3	**Vanilla**	290	38.75 (2.53)	12	10	10	8	12	18	4	18	15	18	12	**25**	18	15	10	12	12	18	18	**25**
4	**EA-tree**	243	35.25 (2.36)	15	12	**25**	12	10	15	15	15	4	6	8	6	15	6	18	15	15	15	12	4
5	**EA-bandit-shift**	219	32.00 (2.28)	8	**25**	4	**25**	4	6	8	4	**25**	15	**25**	10	8	4	8	10	8	6	6	10
6	**EA-bandit**	206	36.35 (2.52)	10	4	8	6	15	10	6	12	12	10	4	12	12	8	**25**	8	10	12	10	12
7	**EA-bandit-tree**	174	34.65 (2.44)	4	18	6	4	6	12	**25**	8	6	8	15	4	10	12	4	6	6	10	4	6
8	**EA-bandit-tree-shift**	162	33.40 (2.17)	6	6	12	10	8	4	10	6	8	4	6	18	4	10	15	4	4	4	8	15

Table II: Configuration 5-10. Rankings table for part 1 algorithms across all games. In this order, the table shows the rank of the algorithms, their name, total F1 points, average of victories and F1 points achieved on each game.

69% (*Vanilla*, 1-6) to 98% (10-14). However, in game 50 (*Intersection*), *EA-bandit* (1-6) is one of the only algorithms achieving a win rate under 100% (only 89%, the others being *Vanilla*, 91% and *EA-bandit-shift*, 99%).

The worst bandit hybrid is *EA-bandit-shift*, achieving only 32.05% win rate in even the best configuration tested. Bandit mutation is generally most beneficial in larger configurations.

2) EA-tree: Across all games, *EA-tree* is better than *Vanilla* in the lower half of the configurations tested. In an overall view of all algorithms and configurations, the tree hybrids rank mid-table, outperforming *EA-bandit*, but not *EA-shift*.

There are several games in which *EA-tree* is significantly better than *Vanilla* in either wins or score, although this effect is mostly observed in low configurations, such as game 25 (*Crossfire*), and game 13 (*Butterflies*). However, in game 36 (*Escape*) *EA-tree* is significantly worse than all other algorithms in win rate, across all configurations.

The worst tree hybrids are *EA-bandit-tree* and *EA-bandit-tree-shift*, both being very close in performance on configuration 10-14, while *EA-tree* ranks second. The statistical tree appears to be most beneficial in low configurations.

3) EA-shift: The shift buffer is the best enhancement analyzed. It outperforms *Vanilla* in all configurations, ranking first in all but 1-6, where tree hybrids are better. Overall, the shift buffer hybrids are good at achieving significantly higher scores than all others. As many games rely on this aspect, the win rates also increase, although not as dramatically.

For example, in game 29 (*Dig Dug*), *EA-shift* (10-14) is significantly better in score than all other algorithms in all configurations. In game 49 (*Infection*), even though it is again significantly better than all others in scores (10-14), its win rate is significantly worse than most others. Nevertheless, in games 91 (*Wait for Breakfast*, 10-14) and 0 (*Aliens*, 1-6), *EA-shift* and its hybrids see a significant increase in both win rates and scores (from 86% to 100% win rate in Aliens, $p \ll 0.001$).

The worst shift buffer hybrids are *EA-bandit-shift* and *EA-bandit-tree-shit*, which achieve a much lower win rate. This leads to the conclusion that combining bandit-mutation with a shift buffer (possibly due to the old information stored by the bandits) is not favorable in this setting.

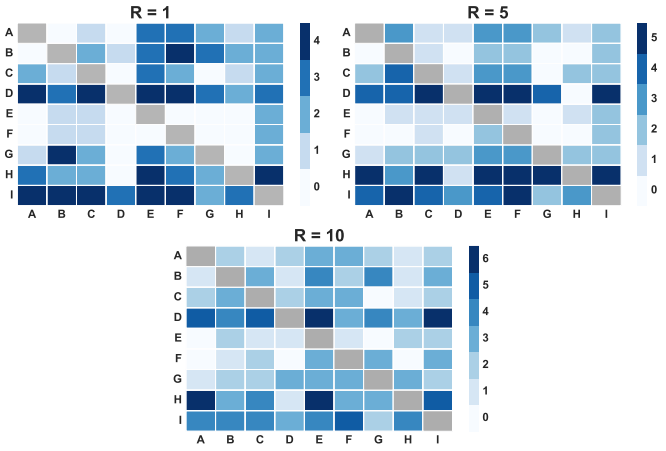

Figure 3: Win percentage for configuration 10-14. The color bar denotes in how many unique games row was significantly better than column. Legend: A = Vanilla, B = EA-roll, C = EA-shift, D = EA-shift-roll, E = EA-tree, F = EA-tree-roll, G = EA-tree-shift, H = EA-tree-shift-roll, I = MCTS

B. EA-roll and its hybrids

The overall results of the second part of experiments suggest that the shift buffer enhancement is even better when combined with rollouts, *EA-shift-roll* being the dominating algorithm, while *Vanilla* and *EA-roll* rank the lowest in most settings. It is interesting to observe the gradual increase in performance of *EA-shift* in all R values, being last out of the shift hybrids in 1-6, but moving up in the rankings with the increase in core parameters. Rollouts seem most advantageous in low configurations, as they become too expensive to compute in the limited budget when individual length grows.

EA-tree-roll performs the worst out of the tree hybrids in all configurations and R values, indicating that the deeper look into the future provided by the rollouts does not have a positive impact on the tree statistics. The best tree hybrid is *EA-tree-shift-roll*, surpassing the variant without rollouts.

Figure 3 presents the significant wins of all variants in configuration 10-14, with the different repetitions $R = \{1, 5, 10\}$. MCTS is included for comparison as the last row/column. It

Config.	R	Best By F1 Points		Best By Win Rate	
		Algorithm	Avg. Wins	Algorithm	Avg. Wins
1-6	1	**EA-shift-roll**	38.35 (2.31)	**EA-tree-shift-roll**	38.60 (2.55)
	5	**EA-shift-roll**	40.10 (2.51)	**EA-shift-roll**	40.10 (2.51)
	10	**EA-shift-roll**	39.35 (2.64)	**EA-shift-roll**	39.35 (2.64)
2-8	1	**EA-shift-roll**	40.35 (2.63)	**EA-shift-roll**	40.35 (2.63)
	5	**EA-shift-roll**	40.75 (2.46)	**EA-shift-roll**	40.75 (2.46)
	10	**EA-shift-roll**	40.20 (2.30)	**EA-shift-roll**	40.20 (2.30)
5-10	1	**EA-shift-roll**	43.20 (2.43)	**EA-shift-roll**	43.20 (2.43)
	5	**EA-shift**	40.05 (2.50)	**EA-shift-roll**	41.85 (2.42)
	10	**EA-shift**	40.05 (2.50)	**EA-shift**	40.05 (2.50)
10-14	1	**EA-shift**	39.75 (2.54)	**EA-shift-roll**	42.80 (2.44)
	5	**EA-shift-roll**	42.05 (2.48)	**EA-tree-shift-roll**	42.70 (2.41)
	10	**EA-shift-roll**	42.35 (2.53)	**EA-shift-roll**	42.35 (2.53)

Table III: The best algorithms (by Formula-1 points and win rate) in all configurations and rollout repetitions (R), as compared against the other variants in the same configuration and the same R value (includes variants without rollouts).

is interesting to note that *EA-shift-roll* is significantly better than most other algorithms in all R values, matching the performance of *MCTS*, but the most in R = 5, then decreasing in R = 10. This suggests that the ideal value peaks in the vicinity of 5. *EA-tree* and *EA-tree-roll* also stand out as the worst algorithms in all R variations tested.

The good performance of EA-shift-roll is also highlighted in Table III, which summarizes the best algorithm in each configuration and R value by both Formula-1 points and win rate. The specific amount of points are not presented due to their high dependence on the other algorithms in the rankings and point distribution, therefore not being comparable independently. *EA-shift-roll* stands out as dominating most settings by both F1 points and win rate, with few exceptions.

One of the interesting games to look at in more detail is game 22 (*Chopper*), *EA-shift-roll* is significantly better in both win rate and scores than most other algorithms in all configurations and R values, the highest win rate being 54% in 1-6, $R = 5$, compared to 35% maximum for *EA-roll* (10-14, $R = 10$). The high improvement in low configurations is of specific interest, as it allows more thinking time in other parts of the evolutionary process for more complex computations.

C. Comparison with MCTS

Finally, we carried out a comparison with MCTS, the dominant technique in GVGAI. Overall, only few of the RHEA variants succeed in significantly outperforming MCTS. However, Table IV shows the direct contrast between the best RHEA variant found during these experiments in terms of generality (thus highest F1 points in individual juxtaposition against the other algorithms), *EA-shift-roll* (10-14, $R = 5$) and MCTS (with a comparable rollout length of 14). Highlighted are the games in which one algorithm is better than the other (even if the difference is not significant).

EA-shift-roll matches the generality of MCTS, achieving the same amount of F1 points, but a higher win rate. When looking at individual games, it becomes clear that this RHEA variant is significantly better than MCTS in 5 games for win rate and 6 games for scores, while being significantly worse in 3 and 6 games, respectively. For example, in game 4, MCTS achieves a win rate of 6%, while *EA-shift-roll* obtains 19% ($p = 0.003$).

Table V shows the comparison between the RHEA variant considered most similar to MCTS (*EA-tree-roll* with $R = 1$) in its best configuration, 10-14, and MCTS (with a rollout length of 14). The fact that the tree is updated passively alongside the RHEA population and it is only used at the end of the evolutionary process to select which action to play leads to a lower significantly lower performance than MCTS in 5 games for win rate and 11 games for score.

However, there are several games where *EA-tree-roll* is significantly better in terms of win rate: game 36 ($p = 0.012$), a game in which EAs traditionally do better than tree search, and game 91 ($p \ll 0.001$).

VII. Conclusions and Future Work

This paper studied the effects of four different enhancements applied to the vanilla version of the Rolling Horizon Evolutionary Algorithm (RHEA), aiming to provide a fair comparison between the methods and identify possible synergies. They were analyzed in four different parameter configurations, with the same general heuristic and in the same set of 20 games of the General Video Game AI (GVGAI) corpus.

The experiments were divided into two parts due to the large scale of the analysis. First, three of the enhancements were tested individually and in all combinations, resulting in 8 algorithms. A bandit system was used to guide mutation (*EA-bandit*); a statistical tree was kept alongside evolution employed in selecting actions at the end of the evolution (*EA-tree*); and a population shifting method was used to carry forward information from one game step to the next (*EA-shift*). Combinations of these methods resulted in interesting hybrids.

The results indicate that the uni-variate bandit system does not work well in this setting where individuals are sequences of actions. This is thought to be due to epistasis: changing one gene in an individual impacts all the subsequent genes as well, therefore the statistics used by the bandits are much less useful. This leads to a line of future work in employing an N-tuple bandit mutation [20] in order to account for the connections between genes. The bandit systems do work better in high configurations, due to fewer evolution iterations, therefore the effect is less pronounced. *EA-shift* and *EA-tree-shift* stood out as the best algorithms in this first part, followed shortly by *EA-tree*. It was observed that the stats tree was more beneficial in small configurations, due to information in small individuals being more accurate than in longer ones. Whereas the shift buffer enhancement led to a significant increase in score gain, as well as raising the win rates in small configurations so as to be similar to those of the vanilla version with large core parameter values. The shift buffer worked so well because reusing information from previous game steps means learning more about the environment in the limited budget available.

The second part of the experiments took the 4 best algorithms found, and added the fourth enhancement (Monte Carlo type rollouts at the end of the individual evaluation, repeated $R = \{1, 5, 10\}$ times) to create 4 new variants (named *EA-roll*

#	Algorithm	Points	Avg. Wins	G-0	G-4	G-13	G-15	G-18	G-22	G-25	G-29	G-36	G-46	G-49	G-50	G-58	G-60	G-61	G-67	G-75	G-77	G-84	G-91
1	**EA-shift-roll**	430	42.05 (2.48)	18	**25**	18	**25**	18	18	18	**25**	**25**	**25**	18	**25**	18	**25**	**25**	**25**	18	18	18	**25**
2	**MCTS**	430	41.30 (1.76)	**25**	18	**25**	18	**25**	**25**	**25**	18	18	18	**25**	18	**25**	18	18	18	**25**	**25**	**25**	18

Table IV: Configuration 10-14, $R = 5$. Best algorithm found compared with MCTS. In this order, the table shows the rank of the algorithms, their name, total F1 points, average of victories and F1 points achieved on each game.

#	Algorithm	Points	Avg. Wins	G-0	G-4	G-13	G-15	G-18	G-22	G-25	G-29	G-36	G-46	G-49	G-50	G-58	G-60	G-61	G-67	G-75	G-77	G-84	G-91
1	**MCTS**	451	41.30 (1.76)	**25**	18	**25**	18	**25**	**25**	**25**	**25**	18	**25**	**25**	18	**25**	**25**	18	18	**25**	**25**	**25**	18
2	**EA-tree-roll**	409	35.90 (2.27)	18	**25**	18	**25**	18	18	18	18	**25**	18	18	**25**	18	18	**25**	**25**	18	18	18	**25**

Table V: Configuration 10-14, $R = 1$. Algorithm most similar to MCTS compared with MCTS. In this order, the table shows the rank of the algorithms, their name, total F1 points, average of victories and F1 points achieved on each game.

for the vanilla version). No significant difference was observed across the configurations, except for *EA-shift*, which saw an increase in performance proportional to the individual length, surpassing its rollout counterpart. Therefore, the longer the individual, the less beneficial the rollouts become.

EA-shift and *EA-tree-shift-roll* showed a promising performance, but the best algorithm emerging was *EA-shift-roll* (using a shift buffer and rollouts repeated $R = 5$ times, configuration 10-14). This method was compared to Monte Carlo Tree Search (MCTS) for validation and it outperformed MCTS significantly in several games. The algorithm considered most similar to MCTS (*EA-tree-roll*, employing the stats tree and rollouts repeated $R = 1$ times), in its best configuration (10-14), was not as good as initially estimated, and worse than most other RHEA variants in this second part of experiments.

Another line of future work will be expanding this study to a wider range of games, as 20 remains a relatively small sample and possibly not indicative of the true potential of these methods. Additionally, determining the characteristics of the specific games that lead to changes in the performance of particular methods would be an interesting study in itself, which would open the possibility of dynamically tuning and turning this features on or off in order to gain the maximum benefit from each one, depending on the problem at hand.

ACKNOWLEDGMENT

This work was funded by the EPSRC Centre for Doctoral Training in Intelligent Games and Game Intelligence (IGGI) EP/L015846/1.

REFERENCES

[1] V. Mnih et al., "Human-Level Control Through Deep Reinforcement Learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[2] D. Perez-Liebana, S. Samothrakis, J. Togelius, S. M. Lucas, and T. Schaul, "General Video Game AI: Competition, Challenges and Opportunities," in *30th AAAI Conference on Artificial Intelligence*, 2016.

[3] D. Perez-Liebana, S. Samothrakis, J. Togelius, T. Schaul, S. Lucas, A. Couetoux, J. Lee, C.-U. Lim, and T. Thompson, "The 2014 General Video Game Playing Competition," in *IEEE Transactions on Computational Intelligence and AI in Games*, vol. PP, no. 99, 2015, p. 1.

[4] R. D. Gaina, J. Liu, S. M. Lucas, and D. Pérez-Liébana, *Analysis of Vanilla Rolling Horizon Evolution Parameters in General Video Game Playing*. Cham: Springer International Publishing, 2017, pp. 418–434.

[5] R. D. Gaina, S. M. Lucas, and D. P. Liébana, "Population Seeding Techniques for Rolling Horizon Evolution in General Video Game Playing," in *Proc. of the Congress on Evolutionary Computation*, 2017.

[6] D. Perez-Liebana, S. Samothrakis, S. M. Lucas, and P. Rolfshagen, "Rolling Horizon Evolution versus Tree Search for Navigation in Single-Player Real-Time Games," in *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, 2013, pp. 351–358.

[7] S. Samothrakis, S. A. Roberts, D. Perez, and S. Lucas, "Rolling Horizon methods for Games with Continuous States and Actions," *Proc. of the Conference on Computational Intelligence and Games (CIG)*, Aug 2014.

[8] S. M. Lucas, S. Samothrakis, and D. Perez, "Fast Evolutionary Adaptation for Monte Carlo Tree Search," in *EvoGames*, 2014.

[9] D. Perez, S. Samothrakis, and S. M. Lucas, "Knowledge-based Fast Evolutionary MCTS for General Video Game Playing," in *IEEE Conference on Computational Intelligence and Games*, 2014, pp. 1–8.

[10] D. Perez-Liebana, J. Dieskau, M. Hünermund, S. Mostaghim, and S. M. Lucas, "Open Loop Search for General Video Game Playing," in *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, 2015, pp. 337–344.

[11] A. J. C. Gittins and J. C. Gittins, "Bandit Processes and Dynamic Allocation Indices," *Journal of the Royal Statistical Society, Series B*, pp. 148–177, 1979.

[12] E. J. Powley, D. Whitehouse, and P. I. Cowling, "Bandits All the Way Down: UCB1 as a Simulation Policy in Monte Carlo Tree Search," in *IEEE Conf. on Computational Intelligence and Games*, 2013, pp. 1–8.

[13] J. Liu, D. Perez-Liebana, and S. M. Lucas, "Bandit-Based Random Mutation Hill-Climbing," in *Proceedings of the Congress on Evolutionary Computation*, 2017. [Online]. Available: http://arxiv.org/abs/1606.06041

[14] J. Liu, J. Togelius, S. M. Lucas, and D. P. Liébana, "Evolving Game Skill-Depth using General Video Game AI Agents," in *Proceedings of the Congress on Evolutionary Computation*, 2017.

[15] H. Horn, V. Volz, D. Perez-Liebana, and M. Preuss, "MCTS/EA Hybrid GVGAI Players and Game Difficulty Estimation," in *Proceedings of the IEEE Conf. on Computational intelligence and Games (CIG)*, 2016.

[16] D. A. Berry and B. Fristedt, *Bandit Problems: Sequential Allocation of Experiments (Monographs on Statistics and Applied Probability), Population and Community Biology* , 1st ed. Springer, Oct. 1985.

[17] C. Browne, E. Powley, D. Whitehouse, S. Lucas, P. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton, "A Survey of Monte Carlo Tree Search Methods," in *IEEE Trans. on Computational Intelligence and AI in Games*, vol. 4, no. 1, 2014, pp. 1–43.

[18] M. J. Nelson, "Investigating Vanilla MCTS Scaling on the GVG-AI Game Corpus," in *Proceedings of the 2016 IEEE Conference on Computational Intelligence and Games*, 2016.

[19] P. Bontrager, A. Khalifa, A. Mendes, and J. Togelius, "Matching Games and Algorithms for General Video Game Playing," in *12th AI and Interactive Digital Entertainment Conf.*, 2016, pp. 122–128.

[20] K. Kunanusont, R. D. Gaina, J. Liu, D. P. Liébana, and S. M. Lucas, "The N-Tuple Bandit Evolutionary Algorithm for Game Improvement," in *Proceedings of the Congress on Evolutionary Computation*, 2017.