Tackling Sparse Rewards in Real-Time Games with Statistical Forward Planning Methods

Raluca D. Gaina, Simon M. Lucas, Diego Pérez-Liébana
Queen Mary University of London, UK
{r.d.gaina, simon.lucas, diego.perez}@qmul.ac.uk

Abstract

One of the issues general AI game players are required to deal with is the different reward systems in the variety of games they are expected to be able to play at a high level. Some games may present plentiful rewards which the agents can use to guide their search for the best solution, whereas others feature sparse reward landscapes that provide little information to the agents. The work presented in this paper focuses on the latter case, which most agents struggle with. Thus, modifications are proposed for two algorithms, Monte Carlo Tree Search and Rolling Horizon Evolutionary Algorithms, aiming at improving performance in this type of games while maintaining overall win rate across those where rewards are plentiful. Results show that longer rollouts and individual lengths, either fixed or responsive to changes in fitness landscape features, lead to a boost of performance in the games during testing without being detrimental to non-sparse reward scenarios.

1 Introduction

When testing Artificial Intelligence agents on multiple distinct games in a general game playing black box setting, the main difficulty the players face is being able to correctly judge and differentiate situations. Most games cannot be fully explored until the end due to their complexity in action space, state space or both.

If the algorithm is targeted at a particular game, human knowledge about the problem can be integrated into the heuristic function in order to effectively guide the search in the right direction, even if no "natural" rewards (from the game) are observed by the agent. However, the lack of domain knowledge in general video game playing poses a significant challenge on how to bias or guide the search effectively in the case of a mostly flat reward landscape (Perez, Samothrakis, and Lucas 2014).

The main goal of this paper is to analyze potential improvements on agents for sparse reward games, but still keep a similar overall performance in those games where rewards are found more often. We study this in the context of Statistical Forward Planning algorithms: Monte Carlo Tree Search and Rolling Horizon Evolution. These offer high performance and rapid adaptation for general video game playing.

In this paper we propose modifications that alter how far the agents can simulate into the future in two different ways. First, we increase the length of the simulations the algorithms are allowed to make (while stretching the budget per game tick accordingly) to test if the agents are able to solve sparse rewards problems better when they are able to see further ahead. And second, we dynamically vary the length of the simulations within a predefined budget per game tick, depending on the flatness of the reward landscape, in order to examine the algorithm's ability to adapt to the various types of problems proposed.

Two base methods are evaluated on 20 real-time games in the General Video Game AI Framework (GVGAI), Monte Carlo Tree Search (MCTS) and Rolling Horizon Evolutionary Algorithms (RHEA), which have recently proven most competitive in this domain. Additionally, all algorithms and variations are evaluated on 5 further deceptive games, the same analysed in (Anderson et al. 2018).

2 Literature Review

The problem tackled in this paper refers to the variety of reward landscapes in games and how most current general methods are not equipped to handle this. Anderson et al. (Anderson et al. 2018) highlight deceptive reward systems in games (i.e. by introducing score gains which guide the AI player away from winning the game), using agents from the General Video Game AI Competition (GVGAI) to show that AI game players can be easily tricked into not finding the optimal solution. Companez et al. (Companez and Aleti 2016) look at enhancements for Monte Carlo Tree Search in Tic-Tac-Toe variations meant to overcome such deceptive issues, highlighting a particular situation where the agent should be able to self-sacrifice in the short run in order to obtain a larger gain in the long run.

The variety of games that general algorithms are expected to achieve a high performance on is noted by Horn et al. in (Horn et al. 2016). They look at the 2D grid-physics games in the GVGAI Framework and identify the different strengths and weaknesses of Evolutionary Algorithms as opposed to Tree Search based methods. The authors propose a game difficulty estimation scheme based on several observable game characteristics, which could be used as a guideline to predict agent performance depending on the game type. Some of the metrics they extracted tie in to the fitness values identified by the algorithms, such as puzzle elements or enemy (possibly random) Non-Player Characters (NPC) which may negatively impact state value estimation. They also observe the lower performance of most algorithms on

sparse reward games, but their study is limited in terms of overcoming the issues highlighted.

Different authors use macro actions to explore the space in physics based games, where one single action may not have much effect on the environment (Perez-Liebana, Rohlfshagen, and Lucas 2012; Liébana et al. 2017). Simply repeating the same action M times (similar to the concept of frame skipping in Reinforcement Learning) proved very effective in the Physical Traveling Salesman Problem (Perez-Liebana, Rohlfshagen, and Lucas 2012), but it did not work in all physics based games tested in the GVGAI Framework (Liébana et al. 2017) due to the coarseness resultant, indicating that a dynamic approach may be better.

One approach to deal with sparse reward landscapes specifically is presented in (Gaina et al. 2017a). Vodopivec describes the use of dynamic rollout increase, proportional to the iteration number, and weighted rollouts in his Monte Carlo Tree Search (MCTS) based entry in the 2016 GVGAI Two-Player track. The purpose of this addition is specified as combining quick reaction to immediate threats with better exploration of areas farther away, if time budget allows. This is an interesting general approach, but computation time is potentially wasted if there are no close rewards to guide the search before rollouts become long enough to retrieve interesting information.

Another approach (Guo et al. 2016) is to combine Deep and Reinforcement Learning on Atari games to learn, from the reward landscape, a bonus function that modifies the UCT policy on MCTS. The authors showed that it's possible to learn from raw perception and improve the performance of MCTS agents in some of these games, by using policy-gradient for reward design.

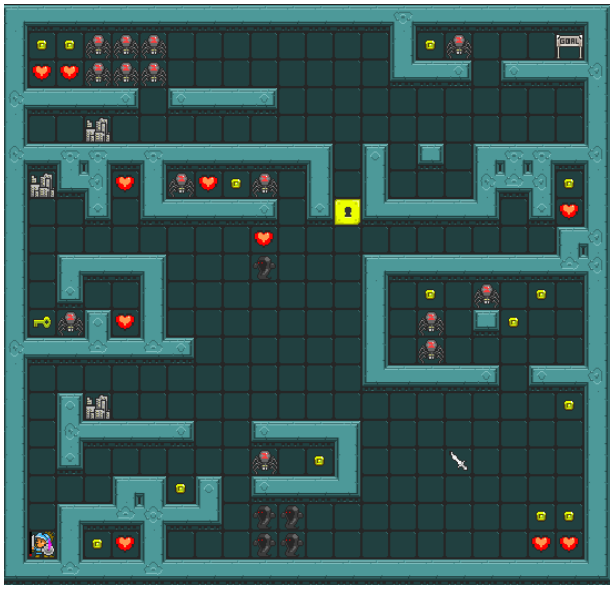
Finally, a complementary set of methods, often referred to as intrinsic motivation, encourage exploration in ways that ignore rewards and focus instead on properties of the state space (or state-action space). The aim is to encourage the agent to explore novel or less visited parts of the state space, or areas that maximise the agent's affordances (Guckelsberger, Salge, and Colton 2016). For non-trivial games, most possible states are never visited due to the vast state space, so statistical feature-based approximations can be used to estimate the novelty of a state (Bellemare et al. 2016). The rollout length adaptation method described here may complement intrinsic motivation methods, but this has not been investigated yet.

3 Background

This section gives background information on the framework (General Video Game AI) and base methods (Rolling Horizon Evolutionary Algorithms and Monte Carlo Tree Search) used in the experiments.

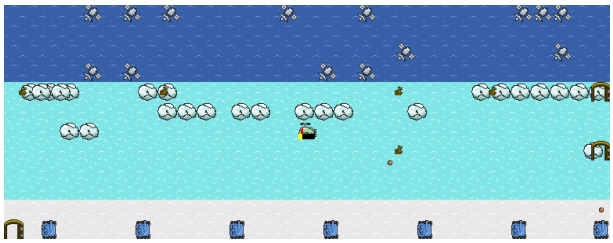
3.1 General Video Game AI Competition Games

The General Video Game AI Framework and Competition (GVGAI) (Perez-Liebana et al. 2015; Gaina, Perez-Liebana, and Lucas 2016) offers various challenges within the field of General Video Game Playing (Levine et al. 2013). There are currently over 160 2D grid-based games in the framework,



(a) Roguelike

(b) Butterflies



(c) Chopper

Figure 1: Games in General Video Game AI Framework.

varying from puzzles to shooters to adventure games. All are written in the Video Game Description Language (Schaul 2014) and are differentiable by several features, such as game object types (NPCs, resources), observability (full or partial) or, in the case of two-player games, player relation (cooperative or competitive). Each game consists of a problem to be solved and there are different winning conditions based on the objectives of the game (e.g. reaching the exit/goal, collecting all treasure, killing all monsters). See Figure 1 for examples of games.

The game rules are not available to the game-playing agents, which only have access to an object describing the current game state (offering observations of the world and other information such as the avatar state, game score, game

tick and game winner, if the game has ended). Additionally, agents may simulate future possible states of the games via a Forward Model. However, copying and advancing game states are the most time-expensive actions performed by agents and should be used in such a way to maximise information gain.

A subset of 20 different games is used in this paper, as analysed in (Gaina et al. 2017b). This set of games comprises of a varied selection regarding game difficulty and game features, as well as including 10 deterministic and 10 stochastic games. As in this paper we are focused on exploring fitness landscapes, it is interesting to distinguish between the different reward systems:

1. **Sparse rewards**: Crossfire, Camel Race, Escape, Hungry Birds, Wait for Breakfast, Modality
2. **Dense rewards**: Digdug, Lemmings, Roguelike, Chopper, Chase, Bait, Survive Zombies, Missile Command, Plaque Attack, Infection, Aliens, Butterflies, Intersection, Seaquest

Sparse reward games feature little to no rewards during the entirety of the game. For example, in "Camel Race" the agent competes against 3 other NPC-controlled camels to make it from one end of the level to the other, while avoiding obstacles. The agent is only rewarded 1 point for finishing the race first. In contrast, dense reward games contain an abundance of rewards. For example, in "Aliens" (adaptation of the well known "Space Invaders"), the agent receives points for each alien they kill, as well as for destroying protective bases. Many aliens and bases are present in each level, thus the agents may gather many points and use the reward system to guide their search.

Additionally, the score is not always increasing linearly. In games such as "Lemmings" or "Plaque Attack", the player is more likely to lose points, but still be doing well and able to win, or even having to lose points in order to win. There may also be games in which winning and gaining the most points are two conflicting goals: in "Butterflies", the player gets points by catching butterflies (random NPCs) and wins when all the butterflies have been caught; however, there are also cocoons in the levels, which can spawn more butterflies if the random NPCs collide with them; therefore, the player would get most points by delaying their win, while not allowing for all cocoons to be opened (in which case the game would end in a loss).

3.2 Monte Carlo Tree Search

Monte Carlo Tree Search (MCTS) repeats several iterations during one game tick as depicted in Figure 2, after which it recommends an action to play (in our implementation, this is the most visited action). MCTS begins by navigating the tree using the Upper Confidence Bound applied to Trees (UCT) policy (to balance between exploration and exploitation) until it encounters a node not yet fully expanded. It then adds a new child of this node to the tree and performs a Monte Carlo rollout (randomly sampling actions and simulating game states using the Forward Model) until either the end of the game or the set rollout depth is reached. The final state is evaluated with a heuristic and its value backed up the tree, updating all nodes visited during the iteration. Our implementation does not store game states in the nodes, but only statistics (Q-value, total number of visits and visits per action).

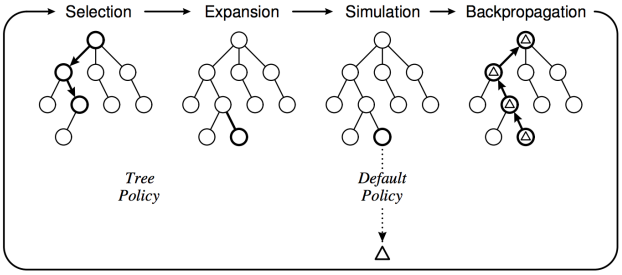


Figure 2: Monte Carlo Tree Search (Browne et al. 2014).

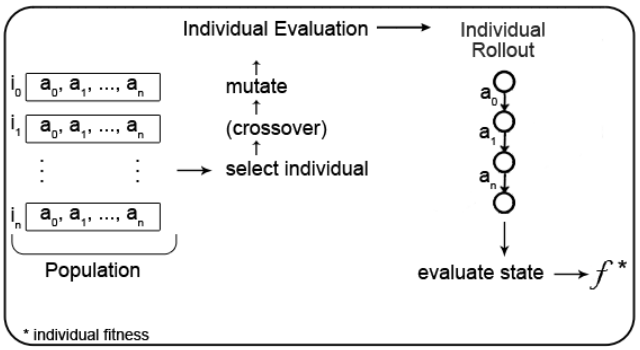


Figure 3: Rolling Horizon Evolutionary Algorithm (Gaina et al. 2017b).

3.3 Rolling Horizon Evolution

Rolling Horizon Evolutionary Algorithm (RHEA) (Gaina et al. 2017b; Gaina, Lucas, and Liébana 2017a; 2017b), first introduced in (Perez-Liebana et al. 2013) is a method used in games as a planning game-playing agent. This technique evolves sequences of actions to be played in the game by iterating over randomly sampled solutions and applying, in our implementation, random mutation to flip one of the actions in the sequence to a new random value; and uniform crossover to combine individuals in interesting ways (after a tournament of size 2 determining the parents involved in crossover). The best individual at each generation is carried forward unchanged through elitism.

The algorithm follows the steps described in Figure 3. RHEA begins by initializing a population of individuals (either uniformly at random, or biased (Gaina, Lucas, and Liébana 2017a)), where each individual represents a sequence of actions. All the individuals are evaluated by simulating through the actions, in turn, until the end of the game or the end of the individual is reached. The final state is evaluated with a heuristic, this value being assigned as the fitness of the respective individual. This process is repeated within the budget during one game tick. At the end of the evolution, the best action is recommended to be played in the game (here, the first action of the best individual).

It is interesting to note the exploration differences of MCTS and RHEA. In binary reward games (e.g. puzzles), it is often the case that a more precise sequence of actions is needed to solve the problem. MCTS explores nodes close to the root most, due to incrementally building the tree, determining better confidence bounds than for states further away. RHEA spreads exploration across the entire space by evolving whole sequences. Therefore, RHEA is able to better sample the large solution space (and find a good overall solution), while MCTS is focused on finding good solutions close to the root and randomly sampling from there.

4 Baseline Methods

For the experiments presented in this paper, an instance of each of the algorithms described in Section 3 is used. The Rolling Horizon Evolutionary Algorithm employs random initialization and a shift buffer for population management, which refers to keeping the population between game ticks instead of discarding it and reinitializing; the first action of all individuals is removed, all other actions shifted one position to the left and a new random action is added at the end. Additionally, Monte Carlo rollouts are added at the end of the traditional individual evaluation. This is the best RHEA variant described in literature (Gaina, Lucas, and Liébana 2017b).

It is important to notice that this work focuses on improving domain-agnostic algorithms on games with sparse rewards. Other agents submitted to the GVGAI Competition (i.e. YOLOBOT (Joppen et al. 2017), the winner of several editions) obtain higher performance than the methods explored here, but also count on stronger heuristics (mostly adapted to respond well to GVGAI games) and combine several algorithms (tree search, A*, best first search, etc.). The focus of this paper is to explore improvements in simpler, game-agnostic algorithms taking their vanilla form as baseline to analyze the effects of the proposed modifications.

Initial experiments attempted to add a tree shifting behaviour in MCTS as well, the equivalent of the shift buffer in RHEA (the method already uses Monte Carlo rollouts, therefore the two algorithms are comparable in that sense). However, this enhancement heavily impacted the algorithm's performance in a negative way. As a result, we considered that the comparison would be most fair if both algorithms were at their best. Our experiments feature the sample MCTS as provided in the GVGAI Framework.

Moreover, we applied the same configuration of parameters to both RHEA and MCTS: a population size of 10 for RHEA, rollout length L of 14, budget of 1000 Forward Model (FM) calls[1]. In experiments where extreme rollout lengths are employed (see Section 5.1), we still evaluate 40 individuals for RHEA and 40 iterations for MCTS by increasing the FM call budget accordingly.

[1]Forward Model calls were used instead of the typical time budget in GVGAI for two reasons. First, it would ensure consistency in results irrespective of the machine used to run the experiments. Second, it would make our results comparable with previous literature employing similar budget constraints.

Both baseline algorithms make use of the same state evaluation function to determine the value of an action (or sequence of actions). This function is shown in Equation 14, where H^+ is a large positive integer and H^- is a large negative integer, both surpassing any rewards the agents may receive from any game. What this translates to is that agents will greatly favour winning (and avoiding a loss, respectively), but they will attempt to maximize the current game score if the game state reached is not final, in order to guide their search. The heuristic function is kept intentionally simple and general in order to focus results on the variations within the algorithms' decision making process.

$$f = score + \begin{cases} H^+, \, if \; win \\ H^-, \, if \; loss \end{cases} \quad (1)$$

5 Experiments

This section details the two different modifications: an increase in the method's lookahead (with appropriately increased budgets) and dynamic changes in its lookahead (when constrained to a set budget).

5.1 Extreme Length Rollouts

This experiment is not feasible in real-time in current regular machines, but as technology advances quickly, the computational power increases as well. Therefore, it is worth exploring whether longer rollouts do produce better results, given an appropriately increased budget as well to keep evaluating 40 individuals for RHEA and 40 iterations for MCTS, as is the case in the default parameter configuration with 1000 FM rollout budget. The longest length previously explored was 24 in (Gaina et al. 2017b), therefore we are considering up to 4 times this length (see Table 1 for details on specific lengths L, their associated budgets are $L \times 60$).

One could expect agents employing extreme length rollouts to spot rewards farther ahead more easily and create better plans to reach said rewards, therefore increasing performance in games with sparse rewards or distant goals. The agents may exhibit poor performance in quick reaction games, as they may ignore immediate threats or rewards and instead focus on longer term goals.

5.2 Dynamic Length Rollouts

This experiment will look instead at dynamically adjusting the length of the rollouts within the 1000 FM calls budget (therefore feasible in real time). The objective is to obtain a more interesting behaviour comprised of quick reactions in situations where rewards are plentiful, and more exploratory longer lookaheads when rewards are sparse.

The pseudocode of the method used to adjust the rollout length is depicted in Algorithm 1. The adjustment is set to occur with a frequency $\omega = 15$ game ticks (Line 1). The feature used to determine a change in rollout length is the flatness of the fitness landscape observed in the previous game tick (f_{Ld}); this is therefore ignored if the first game tick is currently observed (therefore no fitness landscapes were previously recorded; Lines 2-3). The fitness landscape is a vector with all fitness values observed in one game tick by any

Algorithm 1 Adjusting rollout length dynamically

Require: t: current game tick
Require: $fitnessLandscape$: the fitness landscape (all fitness values) observed in the previous game tick
Require: f_{Ld}: fitness landscape flatness
Require: L: rollout length
Require: ω: adjustment frequency
Require: SD_{-}: lower f_{Ld} limit for L increase
Require: SD_{+}: upper f_{Ld} limit for L decrease
Require: M_D: rollout length modifier
Require: MIN_D: minimum value for L
Require: MAX_D: maximum value for L

1: **if** $t \bmod \omega = 0$ **then**
2: **if** $fitnessLandscape = null$ **then**
3: $f_{Ld} \leftarrow SD_{-}$
4: **else**
5: $f_{Ld} \leftarrow \delta(fitnessLandscape)$ ▷ get standard deviation
6: **if** $f_{Ld} < SD_{-}$ **then**
7: $L \leftarrow L + M_D$
8: **else if** $f_{Ld} > SD_{+}$ **then**
9: $L \leftarrow L - M_D$
10: BOUND(L, MIN_D, MAX_D)
11: **function** BOUND(L, MIN_D, MAX_D)
12: **if** $L < MIN_D$ **then**
13: $L \leftarrow MIN_D$
14: **else if** $L > MAX_D$ **then**
15: $L \leftarrow MAX_D$
return L

individual in the population (RHEA) or any rollout (MCTS), and its *flatness* is calculated as the standard deviation (δ) of all the elements of this vector (Line 5).

The length L is then increased by the depth modifier M_D = 5 if f_{Ld} falls below the lower limit (SD_{-} = 0.05), or is decreased by M_D if f_{Ld} is above the upper limit (SD_{+} = 0.4) (see Lines 6-9). The length is capped to always stay between a minimum (1) and a maximum (half of the maximum number of FM calls; Line 10). This translates to shorter rollouts when the fitness values observed are highly varied (therefore more sampling and processing of the current situation is needed to determine the right course of action) and longer rollouts when the fitness landscape is flat, to encourage exploration of solutions farther ahead which would give the agent more information to judge which would be the best move. The values for the different variables (ω, SD_{-}, SD_{+}) were manually tuned for best performance of both algorithms on a random subset of 5 games.

One can reasonably expect dynamic rollouts to improve the overall performance, as agents could possibly adapt better to different situations requiring distinct skills.

6 Results and Discussions

The results reported in this section mainly focus on the win rate achieved by the algorithms. Each method played 100 runs per game, 20 in each of the 5 levels [2]. Fisher's exact test is used to test the significance of win rate differences.

Alg	L	Sparse	Dense	Overall
RHEA	14	25.59 (2.82)	58.04 (1.73)	48.31 (2.05)
	50	29.80 (3.30)	61.33 (1.66)	51.80 (2.14)
	100	36.36 (3.59)	61.04 (1.87)	53.55 (2.37)
	150	36.03 (3.59)	62.63 (2.10)	54.60 (2.53)
	200	37.04 (3.85)	60.89 (1.93)	53.70 (2.49)
MCTS	14	6.90 (1.79)	54.76 (1.65)	40.40 (1.69)
	50	14.31 (3.29)	53.54 (2.08)	41.70 (2.42)
	100	22.39 (3.79)	54.18 (1.58)	44.50 (2.23)
	150	26.77 (3.94)	53.75 (1.49)	45.60 (2.21)
	200	30.98 (4.14)	53.61 (1.52)	46.70 (2.29)

Table 1: Average win rate for long rollouts variations. Distinction is made between sparse and dense rewards games, with the final column averaging over all games. Budget for each algorithm is $L \times 60$.

6.1 Extreme Length Rollouts

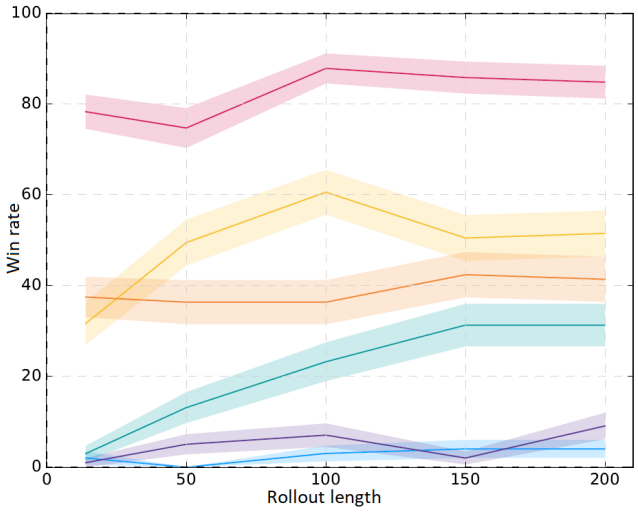
Overall, results suggest that the general trend is the longer the rollouts, the better. However, there is a point where the improvement halts in RHEA (see last column in Table 1). When looking at the different reward systems, the improvement is only noticeable in sparse reward games, whereas the performance in incremental games remains fairly constant; one exception is "Bait" which increases significantly from 16.5% to 37.4% for RHEA with L = 150, $p \ll 0.001$ (this game is a special incremental scoring system game case featuring puzzle elements; see Section 3.1 for games split by score system). A similar trend is observed for MCTS: significant improvement in win rate in sparse reward games (from 14.31% to 30.98%, $p \ll 0.001$), while performance in dense reward games remains constant; thus the performance gain in sparse reward games is not detrimental to the rest of the problems. However, we do notice a striking drop in performance for MCTS in the incremental game "Chopper", where the algorithm falls from 100% win rate to 4% with L = 200; the same is not observed in RHEA, suggesting MCTS to be worse at dealing with immediate threats when considering farther ahead rewards.

Figure 4 shows the win rate of both RHEA and MCTS variants with long rollouts in the sparse reward games. It is interesting to observe that in the game "Escape" both methods increase their performance until they peak (at L=100 for RHEA and L=150 for MCTS), following which the winrate drops again. In most other games we see a steady increase as rollout length goes up. This could suggest that the rollout length should not be pushed to too high values and instead more carefully considered based on the problem at hand.

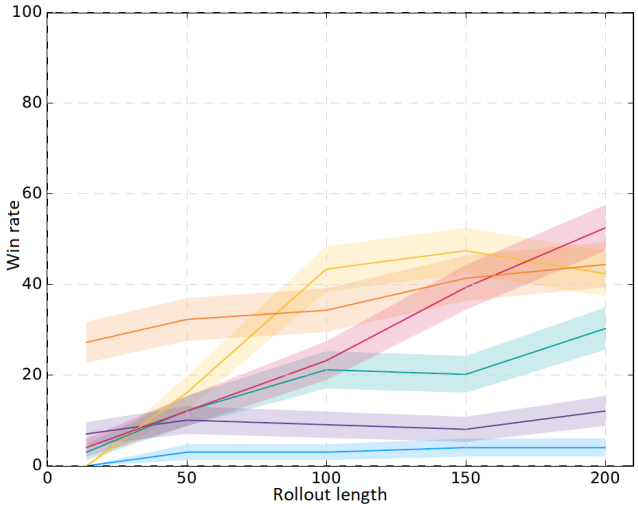
6.2 Dynamic Length Rollouts

The two algorithms tested in this study show very different reactions to dynamic variations of their rollout length.

[2] Full result files can be found in a GitHub repository at: github.com/rdgain/ExperimentData/tree/SparseRewards



(a) RHEA



(b) MCTS

Figure 4: Win rate in sparse rewards games for RHEA (top) and MCTS (bottom) with extreme length rollouts. Shadowed areas indicate the standard error.

This adjustment halves win rate in RHEA (from 48.60% to 21.01%, $p \ll 0.001$), but it improves performance in MCTS, from 40.40% to 44.00% overall, $p = 0.022$.

The explanation for the large drop in win rate suffered by RHEA is the use of the shift buffer. In fact, it is reasonable that altering previously evolved sequences of actions by cutting or increasing them (with new random actions added at the end) changes the sequence (and importantly, the phenotype) too much for the algorithm to be able to handle. This theory was tested and it showed that, by removing the shift buffer, dynamically adjusted rollout lengths in RHEA lead to a 39.55 win rate. This is still lower than the baseline method, but it is at the level of the default MCTS method without dynamic rollouts, suggesting that this adjustment does have the potential of improving performance, with possibly tweaked (or dynamically adjusted as well) parameters.

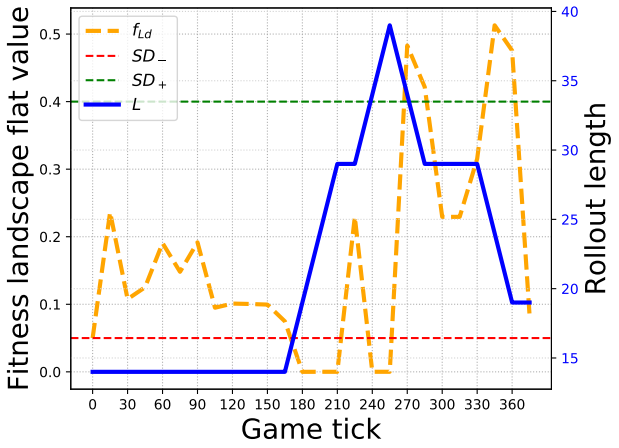


Figure 5: Variations in dynamic rollout length (blue) and fitness landscape flatness (orange) for RHEA agent in the game "Butterflies", level 0. Note that the scale for rollout length is on the secondary (right) Y axis.

Table 2 summarises the win rates of the two methods and their variations on the set of 20 games. Looking more in depth at the two types of reward systems paints an interesting picture. The performance of RHEA remains similar in sparse reward games when dynamic rollouts are employed, whereas the noticeable drop in performance comes from the side of dense reward games, notably "Chopper", from 100% to 56.57%, and "Intersection", from 100% to 43.43%. This indicates dynamic rollouts to be harmful for RHEA in environments with immediate rewards. RHEA tends to shorten the rollout length in such games (see Algorithm 1) and that has been shown to reduce its performance (as seen in (Gaina et al. 2017b)); we hypothesize the drop in win rate to be most likely due to sequences becoming too short for RHEA to be able to cope with.

However, MCTS sees a similar story as in the case of extreme length rollouts: the performance in sparse reward games is significantly improved (from 6.90% to 19.70%), with no detriment to the rest in the set. Some notable examples here are "Escape", which sees an increase in win rate from 0% to 29.29%, and "Wait for Breakfast", from 4% to 42.42%. This suggests dynamic rollouts to be greatly beneficial to MCTS in sparse rewards landscapes.

Figure 5 shows an example of how RHEA varies its rollout length L in the game "Butterflies" and the corresponding fitness landscape flatness f_{Ld}. The upper and lower limits (SD_+ and SD_-, respectively) are the points where the algorithm is expected to adjust its rollout length depending on its assessment of the fitness landscape. It is interesting to note that the rollout length does match the shape of the fitness landscape flatness. The fact that the algorithm reduces its rollout length after a peak at game tick 255 suggests that RHEA is able to successfully use the longer rollouts to ad-

Alg	Sparse	Dense	Overall
RHEA	25.59 (2.82)	58.04 (1.73)	48.31 (2.05)
RHEA-dyn	10.61 (2.61)	25.47 (2.06)	21.01 (2.22)
MCTS	6.90 (1.79)	54.76 (1.65)	40.40 (1.69)
MCTS-dyn	19.70 (3.37)	54.47 (1.89)	44.04 (2.33)

Table 2: Win rates for RHEA and MCTS, vanilla and dynamic variants (non-shift RHEA). Distinction is made between sparse and dense reward systems, with the last column averaging win rates over all games.

just its search and find the more interesting parts of the level to win the game.

6.3 Deceptive Games

The last experiment was to test these methods on the deceptive games presented by Anderson et al. (Anderson et al. 2018). It is expected that the adjusted variants would perform better than the baseline, as they are less biased, have more information or better adapt to various situations, respectively, when making decisions. The most interesting results on the 5 games tested are as follows.

- `decepticoins`: RHEA-dynamic performs significantly better than all other RHEA variations, in both win rate and score (55.56% win rate, a significant 40% improvement over baseline). All MCTS variations achieve a 79.8% win rate, although the extreme rollout length variations complete the games the fastest (200 ticks faster than the baseline on average).
- `flower`: All algorithms achieve 100% win rate, but MCTS with long rollouts is overall significantly better than the baseline in score, with over 200 points improvement for all rollout lengths.
- `invest`: No algorithm manages to solve this game, but all fitness exploratory variations of the algorithms are significantly better than the baseline in score (100-300 point improvement for MCTS, 10-100 points for RHEA).
- `sistersavior`: The win rate in this game is on average very low ($3.03\% \pm 1.08$), with 4 algorithms unable to solve it: baseline MCTS, MCTS-dynamic, RHEA-150 and RHEA-200. The highest win rate is achieved by MCTS-100 ($10.26\% \pm 4.86$), followed closely by RHEA-50 with $7.69\% \pm 4.27$ win rate.
- `waferthinmintsexit`: All algorithms achieve 100% win rate. RHEA-dynamic is significantly better in score than the baseline, 2.68 (± 0.36) to 1.16 (± 0.11) points.

RHEA-dynamic performed much better than the baseline method in most of the GVGAI games tested. There was not much difference observed in some games in terms of win rate, all variations achieving either 100% or 0% victories, although there were overall improvements in either win rate or game score in all cases over the baseline methods. This indicates our modified methods to be more robust to deceptive reward systems.

7 Conclusion

This paper looks at analysing various ways to explore the fitness landscapes in 20 games from the General Video Game AI Framework (GVGAI), for two different algorithms, Monte Carlo Tree Search (MCTS) and Rolling Horizon Evolutionary Algorithm (RHEA). Two experiments are carried out to this extent: increasing the rollout length (to 50, 100, 150 and 200 from the baseline 14) and dynamically adjusting the rollout length based on the flatness of the fitness landscapes, in order to allow for quick reactions in busy environments or more exploration in sparse rewards scenarios. All methods are also tested on a set of human-crafted deceptive reward games to analyze whether their fitness exploration methods lead to better results in such games.

Overall, modified methods are shown to perform better than the baseline methods in sparse reward games, without affecting success rates in dense reward games. One exception is RHEA with dynamic rollouts, which halves win rate from 48.60% to 21.05%. Further analysis into this aspect suggested that this was due to the shift buffer enhancement added to the RHEA variant, which is unable to cope with the change in phenotype between game ticks where the sequence length is varied. By removing the shift buffer, the performance of RHEA with dynamic rollouts becomes comparable to baseline MCTS, at 39.55% win rate.

The algorithms reacted well to the increase in rollout length, their performance increasing with the length in sparse reward games, while performance in dense reward games was kept fairly constant; there were two exceptions to this rule in the games "Butterflies" for both methods and "Chopper" for MCTS only, where increased rollout length is actually detrimental. This is thought to be due to the immediate rewards needed to be collected in this game which may be ignored when the rollout length becomes too large. When the rollout length was dynamically adjusted, RHEA and MCTS reacted differently, RHEA seeing a general decrease in performance in games based on dense reward systems, whereas MCTS saw an increase in performance in sparse reward games. This shows that RHEA is more sensitive to games requiring quick decision making, whereas MCTS benefits from the adjustments which aid in its traditionally poor exploration in binary games.

It is worthwhile mentioning that, although these experiments employ the GVGAI framework, the applicability of the findings extend beyond these games. In particular, this work focuses on modifications to overcome the presence of sparse rewards, an issue present not only in other games such as some in the Atari Learning Environment (Bellemare et al. 2013) and more complex games, but also in other real life scenarios, such as engineering or robotics.

Regarding future work, although this is a very interesting step towards better understanding of agent behaviour, more analysis can be carried out for the various scenarios proposed in this study, including different metrics (game score, GVGAI generality score) or optimizing dynamic rollout adjustment parameters. Additionally, the reactions of the methods to macro-actions in this environment and dynamic length macro-actions could be studied as well. Last but not least, more interesting problems with various features to

their fitness landscapes will be introduced to the methods in order to correctly assess exactly why some algorithms react better to some situations than others.

8 Acknowledgements

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

References

Anderson, D.; Stephenson, M.; Togelius, J.; Salge, C.; Levine, J.; and Renz, J. 2018. Deceptive Games. In *EvoStar*.

Bellemare, M. G.; Naddaf, Y.; Veness, J.; and Bowling, M. 2013. The Arcade Learning Environment: An Evaluation Platform for General Agents. *Journal of Artificial Intelligence Research* 47:253–279.

Bellemare, M.; Srinivasan, S.; Ostrovski, G.; Schaul, T.; Saxton, D.; and Munos, R. 2016. Unifying count-based exploration and intrinsic motivation. In Lee, D. D.; Sugiyama, M.; Luxburg, U. V.; Guyon, I.; and Garnett, R., eds., *Advances in Neural Information Processing Systems 29*. Curran Associates, Inc. 1471–1479.

Browne, C.; Powley, E.; Whitehouse, D.; Lucas, S.; Cowling, P.; Rohlfshagen, P.; Tavener, S.; Perez, D.; Samothrakis, S.; and Colton, S. 2014. A Survey of Monte Carlo Tree Search Methods. 4(1):1–43.

Companez, N., and Aleti, A. 2016. Can monte-carlo tree search learn to sacrifice? *Journal of Heuristics* 22(6):783–813.

Gaina, R. D.; Couëtoux, A.; Soemers, D. J.; Winands, M. H.; Vodopivec, T.; Kirchgessner, F.; Liu, J.; Lucas, S. M.; and Perez-Liebana, D. 2017a. The 2016 two-player gvgai competition. *IEEE Transactions on Computational Intelligence and AI in Games*.

Gaina, R. D.; Liu, J.; Lucas, S. M.; and Liébana, D. P. 2017b. Analysis of Vanilla Rolling Horizon Evolution Parameters in General Video Game Playing. In *Springer Lecture Notes in Computer Science, Applications of Evolutionary Computation, EvoApplications*, number 10199, 418–434.

Gaina, R. D.; Lucas, S. M.; and Liébana, D. P. 2017a. Population Seeding Techniques for Rolling Horizon Evolution in General Video Game Playing. In *Proceedings of the Congress on Evolutionary Computation*.

Gaina, R. D.; Lucas, S. M.; and Liébana, D. P. 2017b. Rolling Horizon Evolution Enhancements in General Video Game Playing. In *Proceedings of IEEE Conference on Computational Intelligence and Games*.

Gaina, R. D.; Perez-Liebana, D.; and Lucas, S. M. 2016. General Video Game for 2 Players: Framework and Competition. In *Proceedings of the IEEE Computer Science and Electronic Engineering Conf.*

Guckelsberger, C.; Salge, C.; and Colton, S. 2016. Intrinsically Motivated General Companion NPCs via Coupled Empowerment Maximisation. In *Proc. of the Conference on Computational Intelligence and Games*.

Guo, X.; Singh, S.; Lewis, R.; and Lee, H. 2016. Deep learning for reward design to improve monte carlo tree search in atari games. *arXiv preprint arXiv:1604.07095*.

Horn, H.; Volz, V.; Pérez-Liébana, D.; and Preuss, M. 2016. MCTS/EA hybrid GVGAI players and game difficulty estimation. In *2016 IEEE Conference on Computational Intelligence and Games (CIG)*, 1–8.

Joppen, T.; Moneke, M.; Schroder, N.; Wirth, C.; and Furnkranz, J. 2017. Informed Hybrid Game Tree Search for General Video Game Playing. *IEEE Transactions on Computational Intelligence and AI in Games*.

Levine, J.; Lucas, S. M.; Mateas, M.; Preuss, M.; Spronck, P.; and Togelius, J. 2013. General Video Game Playing. In *Artificial and Computational Intelligence in Games, Dagstuhl Follow-Ups*, volume 6, 1–7.

Liébana, D. P.; Stephenson, M.; Gaina, R. D.; Renz, J.; and Lucas, S. M. 2017. Introducing Real World Physics and Macro-Actions to General Video Game AI. In *Proceedings of IEEE Conference on Computational Intelligence and Games*.

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

Perez-Liebana, D.; Samothrakis, S.; Togelius, J.; Schaul, T.; Lucas, S.; Couetoux, A.; Lee, J.; Lim, C.-U.; and Thompson, T. 2015. The 2014 General Video Game Playing Competition. In *IEEE Transactions on Computational Intelligence and AI in Games*, volume PP, 1.

Perez-Liebana, D.; Rohlfshagen, P.; and Lucas, S. 2012. The Physical Travelling Salesman Problem: WCCI 2012 Competition. In *IEEE Congress on Evolutionary Computation (CEC)*, 1–8.

Perez, D.; Samothrakis, S.; and Lucas, S. 2014. Knowledge-based fast evolutionary mcts for general video game playing. In *Computational Intelligence and Games (CIG), 2014 IEEE Conference on*, 1–8. IEEE.

Schaul, T. 2014. An Extensible Description Language for Video Games. In *IEEE Transactions on Computational Intelligence and AI in Games*, volume 6, 325–331.