Enhancing Playout Policy Adaptation for General Game Playing

Chiara F. Sironi[1][0000−0001−9795−9653], Tristan Cazenave[2][0000−0003−4669−9374], and Mark H. M. Winands[1][0000−0002−0125−0824]

[1] Game AI & Search Group, Department of Data Science and Knowledge Engineering, Maastricht University, Maastricht, The Netherlands
{c.sironi,m.winands}@maastrichtuniversity.nl

[2] LAMSADE, Université Paris-Dauphine, PSL, CNRS, France
tristan.cazenave@lamsade.dauphine.fr

Abstract. Playout Policy Adaptation (PPA) is a state-of-the-art strategy that controls the playouts in Monte-Carlo Tree Search (MCTS). PPA has been successfully applied to many two-player, sequential-move games. This paper further evaluates this strategy in General Game Playing (GGP) by first reformulating it for simultaneous-move games. Next, it presents two enhancements, which have been previously successfully applied to a related MCTS playout strategy, the Move-Average Sampling Technique (MAST). These enhancements consist in (i) updating the policy for all players proportionally to their payoffs, instead of updating it only for the winner of the playout, and (ii) collecting statistics for N-grams of moves instead of single moves only. Experiments on a heterogeneous set of games show both enhancements to have a positive effect on PPA. Results also show enhanced PPA variants to be competitive with MAST for small search budgets and better for larger ones.

Keywords: Monte-Carlo Tree Search · Playout Policy Adaptation · General Game Playing.

1 Introduction

Monte-Carlo Tree Search (MCTS) [15, 8] is a simulation-based search algorithm that has been successfully applied to various domains [2], one of which is General Game Playing (GGP) [13]. In GGP, the aim is to create agents that are able to play a wide variety of possibly unknown games, only by being given their rules at run-time. The fact that MCTS does not necessarily need domain-specific knowledge is what makes it suitable to be used in GGP.

Many successful enhancements and modifications of MCTS have been investigated [2]. In GGP, particular attention has been given to strategies that can learn how to control the search on-line, i.e. while playing the game, because no domain-specific knowledge is available in advance. Playout Policy Adaptation (PPA) [3] is among the domain-independent strategies that have been proposed to guide the search in MCTS. PPA learns the playout policy on-line by keeping weights for each move visited during the search and adapting them according to the performance of the moves during game simulations. The weights of the moves performed by the winner of a simulation are increased, while the weights of all the legal moves that were available for the winner are decreased. The learned weights are then used in each playout to define a move-selection policy. PPA has been successfully applied to various two-player, sequential-move games [3]. This, together with the fact that it is domain-independent, makes it promising to be further evaluated in GGP.

Moreover, PPA shares some characteristics with another domain-independent playout strategy for MCTS, the Move-Average Sampling Technique (MAST) [11]. Like PPA, MAST collects statistics about the general performance of the available moves in the game, and biases the playout towards moves that have performed overall well so far. The main difference is that MAST learns the value of a move by looking only at its average performance in the game, while PPA learns it by comparing the move with the other available moves for the state in which it has been played. For MAST, various successful modifications have been tested in the literature [11, 23]. The similarity of MAST and PPA, makes such modifications promising to be tested for the latter as well.

The main goal of this paper is to evaluate the PPA algorithm for General Game Playing, looking at possible enhancements that can further improve its performance. First of all, this paper extends the formulation of the PPA algorithm for simultaneous-move games to make it applicable also to such type of games. Moreover, two different modifications of the PPA algorithm are presented. One of them consists in updating the statistics of all players proportionally to their payoff as MAST does, instead of updating only the statistics of the winner by a fixed amount. The other is an enhancement previously tested successfully for MAST [23], the use of statistics collected for N-grams (i.e. sequences) of moves, instead of only for single moves. The PPA strategy and its enhancements are tested on a heterogeneous set of games of the Stanford GGP project [12], of which the rules are represented in GDL (Game Description Language) [16].

The paper is structured as follows. First, Section 2 presents previous work related to PPA. Next, Section 3 gives background knowledge on MCTS, MAST and its enhancements. Subsequently, Sections 4 and 5 describe the PPA algorithm and its enhancements, respectively. Experimental results are discussed in Section 6. Finally, Section 7 gives the conclusions and discusses future work.

2 Related Work

The PPA algorithm seems to be a promising playout strategy for MCTS in many domains. Previous literature already investigated variants of the algorithm that improve its performance on a set of two-player, sequential-move games (e.g. Breakthough, Knightthrough, Domineering, ...). For example, Cazenave [4] proposed to extend PPA with move features (PPAF). The PPAF algorithm learns weights like PPA does, but distinguishes moves depending on some features they are associated with (e.g. whether they are a capture move or a simple movement).

PPAF is harder to apply in GGP, because it would require an on-line analysis of the game rules to first extract relevant move features for a given game that might be previously unknown. Another example of PPA variant that has been investigated is the PPAF strategy with memorization (PPAFM) [5]. With respect to PPAF, the PPAFM strategy does not reset the learned policy between game steps, but keeps it memorized and reuses it. This modification can be easily applied to standard PPA as well.

The interest in the PPA algorithm is motivated also by the success of the closely related Nested Rollout Policy Adaptation (NRPA) [18]. NRPA is designed for single-player games and, similarly to PPA, it learns a playout policy by associating weights to the different available moves. The difference is that NRPA is structured with nested search levels and updates the policy using the best playout found so far at each level. Because the best playout is ill-defined for multi-player games, PPA updates the policy of a given player using the playouts that resulted in a win for such player. NRPA has achieved good results in games, such as Morpion Solitaire and crossword puzzles [18], but has also been applied to solve traveling salesman problems [7, 9] and logistic problems [10], such as vehicle routing, container packing and robot motion planning.

Finally, as already mentioned, PPA is similar to the MAST strategy, which has been successfully applied to GGP [11] and has been subsequently extended to the N-grams Selection Technique (NST) [23]. NST uses statistics of N-grams of moves to guide the playouts and has been shown successful as domain-independent strategy in GGP both for board games [23] as well as for video games [22]. Moreover, Powley et al. [17] used the same idea of NST to implement their strategy, the N-gram-Average Sampling Technique (NAST). This technique has been shown successful also for imperfect-information games [17].

3 Background

This section provides background on MCTS (Subsection 3.1), on the UCT selection strategy (Subsection 3.2), on the MAST playout strategy (Subsection 3.3), and on the NST playout strategy (Subsection 3.4).

3.1 Monte-Carlo Tree Search

MCTS [15, 8] is a simulation based-search strategy that incrementally builds a tree representation of the state space of a game. MCTS performs multiple simulations of the game until a given search budget expires. Each MCTS simulation is divided into the following four phases:

Selection: during this phase, MCTS visits the tree built so far using a *selection strategy* to choose which move to visit in each traversed node. The selection phase ends when a tree node that has not been fully expanded is reached. A node is fully expanded when all its successor nodes have been added to the tree.

Expansion: during this phase, one or more nodes are added to the tree. Such node(s) are selected among the not yet explored children of the last node visited during the selection phase.

Playout: during the playout phase, MCTS starts from a node that was added to the tree during the expansion phase, and simulates the game until a terminal state or a fixed depth is reached. A *playout strategy* is used in each state to select which move to visit. The simplest playout strategy selects moves randomly among the legal ones.

Backpropagation: during this phase, the payoff obtained at the end of the simulation is propagated back through all the visited tree nodes, and used to update statistics about the performed moves. For multi-player games, a tuple of payoffs, one for each player, is propagated back in the tree.

When the search budget expires, MCTS returns one of the moves in the root node to be played in the real game. Commonly, the move that is returned is either the one with the highest number of visits or the one with the highest average payoff. This paper uses the second option.

Note that for simultaneous-move games, MCTS visits in each state a joint move. This paper uses an implementation of MCTS that represents joint moves as vectors formed by one individual move for each player, as described by Sironi [21]. This means that both the selection strategy and the playout strategy have to select a move for each of the players in each visited node or state, respectively.

3.2 Upper Confidence Bounds Applied to Trees

In MCTS, a selection strategy takes care of selecting a move for each moving player in each tree node visited during the search. The standard MCTS selection strategy is UCT (Upper Confidence bounds applied to Trees [15]). Given a state s, UCT chooses the move a^* for a player p using the UCB1 sampling strategy [1] as shown in Equation 1.

$$a^* = \operatorname*{argmax}_{a \in A(s,p)} \left\{ Q(s,a) + C \times \sqrt{\frac{\ln N(s)}{N(s,a)}} \right\} \quad . \tag{1}$$

Here, $A(s,p)$ is the set of legal moves of player p in state s, $Q(s,a)$ is the average payoff of all simulations in which move a has been selected in state s, $N(s)$ is the number of times state s has been visited during the search and $N(s,a)$ is the number of times move a has been selected whenever state s was visited. The C constant is used to control the balance between exploitation of good moves and exploration of less visited ones.

3.3 Move-Average Sampling Technique

A successful playout strategy for MCTS is MAST [11]. The main idea behind MAST is that a move that is good in one state is likely to be good also in other states. During the search, MAST memorizes for each move a of each player p the expected payoff $Q(a,p)$ computed over all the simulations in which move a has been played so far by player p. These statistics are used to select moves during the playout. The original implementation of MAST selects moves in a state s

according to the move probabilities computed using the Gibbs measure reported in Equation 2.

$$Prob(s,a) = \frac{e^{(Q(a,p)/\tau)}}{\sum_{a' \in A(s,p)} e^{(Q(a',p)/\tau)}} \tag{2}$$

Here, a is a legal move for player p, $A(s,p)$ is the set of all the moves that are legal for player p in s, and τ is a parameter that controls the shape of the distribution. High values of τ make the distribution more uniform making it favor more exploration of the moves, while low values stretch it, making it favor more exploitation.

Later research on MAST has shown that the use of an ϵ-greedy strategy to select moves gives a significantly better performance than the use of the Gibbs measure in most of the tested games [23, 17]. The ϵ-greedy strategy chooses the move with highest expected payoff $Q(a,p)$ with probability (1-ϵ) and a random move with probability ϵ.

3.4 N-Gram Selection Technique

The N-grams Selection Technique (NST) [23] is an extension of MAST. Like MAST, NST biases the playouts towards moves that have performed overall well so far. The difference is that NST also exploits information about sequences (N-grams) of consecutive moves. For each considered N-gram of moves up to a given length L, NST memorizes the average payoff obtained by all the simulations in which such N-gram of moves occurred. Note that when L is set to one, NST behaves exactly like MAST, considering statistics for single moves only.

More in detail, after each MCTS simulation, NST extracts from the simulated path all N-grams of moves up to a certain length L and uses the payoff obtained by the simulation to update their expected payoff. Duplicate N-grams in the same simulation are not detected. Therefore, the payoff of an N-gram will be updated for each of its occurrences. When extracting N-grams, NST assumes that the game is simultaneous move and each player plays a move in each turn. The N-grams of each player are updated with the payoff obtained by such player at the end of the simulation. For a game with n players, NST orders the players according to their index, $p = 1, ..., n$. For games specified in GDL, the index of a player corresponds to its order of appearance in the game rules. Assume the notation $a_{(p,t)}$ to indicate the move selected by player p during step t of the simulation. For a given player p, N-grams are extracted from a simulation with T steps as follows:

$$\begin{aligned}
&\text{1-Grams} = \{\langle a_{(p,t)} \rangle | t = 1, ..., T\} \\
&\text{2-Grams} = \{\langle a_{((p-1+n) \bmod n, t-1)}, a_{(p,t)} \rangle | t = 2, ..., T\} \\
&\text{3-Grams} = \{\langle a_{((p-2+n) \bmod n, t-2)}, a_{((p-1+n) \bmod n, t-1)}, a_{(p,t)} \rangle | t = 3, ..., T\} \\
&\qquad\quad \vdots
\end{aligned} \tag{3}$$

This means that N-grams of an arbitrary length $l < L$ for player p are formed by the move of player p at a certain time step t in the simulation, preceded by the moves in the previous $l-1$ steps of the simulation, each performed by the previous $l-1$ players in the cyclic order, respectively.

The statistics collected for the N-grams are used by NST to select moves during the playout. The selection of a move in a state during the playout works as follows. Like for MAST, with probability ϵ a move is chosen randomly. Otherwise, with probability $(1-\epsilon)$ the move with the highest NST value is chosen. Assume s to be the state visited during step t of the simulation, and a_p to be the move of player p for which to compute the NST value. This value is computed as follows:

$$\begin{aligned} NST(a_p) =&(Q(\langle a_p \rangle)+ \\ & Q(\langle a_{((p-1+n) \bmod n, t-1)}, a_p \rangle)+ \\ & Q(\langle a_{((p-2+n) \bmod n, t-2)}, a_{((p-1+n) \bmod n, t-1)}, a_p \rangle)+ \\ & \vdots \\ & Q(\langle a_{((p-L+1+n) \bmod n, t-L+1)}, ..., a_p \rangle)))/L \end{aligned} \tag{4}$$

When computing the NST value of a move, the expected value of the corresponding 1-gram is always considered, while the value of longer N-grams is considered only if the N-gram has been visited more than V times. This ensures that longer N-grams are used only when their value becomes more accurate.

The presented implementation of NST can also be applied to a sequential-move game by considering that players play a pass move (e.g. *noop*) when they are not on their turn. For sequential-move games, the order of players that is used by NST is the same in which the players play their turn. For some GDL games this order corresponds to the order in which roles are reported in the game rules. When this does not happen, NST re-computes the player order for each simulation by looking at which player(s) have legal moves other than *noop* in each turn. Quad is an example of such game, where a player can choose to perform more than one move before giving the control to another player. In this way, there will only be N-grams formed by meaningful moves of each player, and N-grams formed by *noop* moves only. Thus, no N-grams will mix meaningful moves with *noop* moves. This means that the statistics for the N-grams that are formed only by *noop* moves will not influence the selection of other moves.

4 Simultaneous-Move Playout Policy Adaptation

The Playout Policy Adaptation (PPA) strategy [3, 4] is used in MCTS to guide the search during the playout. Similarly to MAST, PPA collects information about the moves visited during the search and uses it to guide future playouts. More precisely, PPA keeps a weight for each possible move of a player, and during the playout it selects moves with a probability proportional to the exponential of their weights. At the start of the search, the weights are initialized to 0. At the end of each playout, the weights of the moves visited by the winning player are increased by a constant α. At the same time, all its legal moves in the visited states are decreased proportionally to the exponential of their weight. The α

Algorithm 1 Pseudocode for PPA

```
procedure PLAYOUT($s, P, W, playout$)
Input: state $s$ from which to start the playout, set $P$ of all the players in the game, matrix $W$ of move weights for each player, $playout$ that contains the states and joint moves visited so far in the current MCTS simulation from the root.
Output: tuple $\boldsymbol{q}$ of payoffs obtained by the players in the terminal state of the current MCTS simulation.
    while not $s$.ISTERMINAL( ) do
        $\boldsymbol{a}^* = \langle a_1^*, ..., a_{|P|}^* \rangle \leftarrow$ empty vector of size $|P|$          ▷ empty joint move
        for $p \leftarrow 1, ..., |P|$ do
            $z \leftarrow 0.0$
            for move $a \in A(s,p)$ do
                $z \leftarrow z + exp(k \times W_{(p,a)})$
            $Prob \leftarrow$ empty probability distribution
            for move $a \in A(s,p)$ do          ▷ creation of probability distribution
                $Prob(a) \leftarrow \frac{exp(k \times W_{(p,a)})}{z}$
            $a_p^* \sim Prob$          ▷ sampling move from distribution
        add $s$ and $\boldsymbol{a}^*$ to the $playout$
        $s \leftarrow$ NEXT($s, \boldsymbol{a}^*$)          ▷ advance to next state
    $w \leftarrow s$.GETWINNERINDEX( )
    if $w \neq$ null then
        $W \leftarrow$ ADAPT($w, W, playout$)          ▷ adapt weights if there is a winner
    $\boldsymbol{q} = \langle q_1, ..., q_{|P|} \rangle \leftarrow s$.GETPAYOFFS( )
    return $\boldsymbol{q}$
```

```
procedure ADAPT($w, W, playout$)
Input: index $w$ of the player that won the playout, matrix $W$ of move weights for each player, $playout$ that contains all the states and joint moves visited so far in the current MCTS simulation from the root.
Output: updated matrix of move weights for each player, $V$.
    $V \leftarrow W$          ▷ copy the weights
    for $i \leftarrow 1, ..., |playout|$ do
        $s \leftarrow playout_i$.GETVISITEDSTATE( )
        $\boldsymbol{a}^* \leftarrow playout_i$.GETVISITEDJOINTMOVE( )
        $z \leftarrow 0.0$
        for move $a \in A(s,w)$ do
            $z \leftarrow z + exp(W_{(w,a)})$
        for move $a \in A(s,w)$ do
            if $a = a_w^*$ then
                $V_{(w,a)} \leftarrow V_{(w,a)} + \alpha$          ▷ increment weight of visited move
            $V_{(w,a)} \leftarrow V_{(w,a)} - \alpha \times \frac{exp(W_{(w,a)})}{z}$          ▷ decrement weight of all legal moves
    return $V$
```

constant represents the learning rate of the algorithm. The algorithm presented in this paper memorizes the weights at the end of each turn to re-use them in the subsequent turn. This choice is motivated by previous literature, which showed that memorizing the playout policy for the PPAF algorithm is beneficial [5].

The pseudocode for the PPA playout strategy for simultaneous-move games is given in Algorithm 1. The algorithm is also applicable to sequential-move games by assuming that players play the *noop* move when it is not their turn. The procedure PLAYOUT$(s, P, W, playout)$ shows how one playout is performed. The procedure requires the state from which to start the playout, s, and the list of players P. Moreover, PPA uses a matrix of weights W, where each entry $W_{(p,a)}$ represents the weight for move a of player p. It also requires a list that memorizes the states and the players' joint moves visited so far during the selection phase of the current MCTS simulation (the *playout* variable in the pseudocode). During the playout, PPA selects a move a^*_p for each player p in each visited state s. To select a move for a player in s it computes a probability distribution over the legal moves proportionally to their weights. Note that the probability of a move is computed using the Gibbs measure presented in Equation 2, where $\tau = \frac{1}{k}$. We keep the notation with k to be consistent with how previous literature introduced the PPA algorithm. Once the moves of all players have been selected, the joint move $\boldsymbol{a}^*$ is used to advance to the next state. At the end of the playout, if there is a single winner, its weights are updated by the ADAPT$(w, W, playout)$ procedure. Finally, a tuple of playoffs, $\boldsymbol{q}$, with an entry for each player, is returned.

The procedure ADAPT$(w, W, playout)$ shows how the weights of the winner are updated at the end of the playout. Other than the index of the winner w and the matrix of weights W, the procedure takes as input the *playout* variable, which now contains the complete list of states and joint moves visited during the current simulation. First of all, the matrix of weights is copied in the matrix V, so that, while V is being updated, the weights of the policy for the previous simulation can still be accessed in W. Subsequently, the algorithm iterates over all the entries in the *playout*. For each visited state s, the weight of the move played by the winner w in s is incremented by α, while the weight of all the legal moves of w in s is decreased proportionally to the exponential of their weight. This means that the sum of all the performed updates in a state is always zero.

5 PPA Enhancements

This section presents the enhancements for PPA that are evaluated in this paper. Both of them have been previously used for MAST [11, 23], and are adapted to be combined with PPA. More precisely, Subsection 5.1 presents the payoff-based update of PPA weights, and Subsection 5.2 explains how PPA can be modified to consider weights for N-grams of moves instead of only for single moves.

5.1 Payoff-Based Update

A characteristic that distinguishes MAST from PPA is that after a simulation, MAST updates the statistics of the moves performed during the simulation for all the players, while PPA updates the weights only of the winner of the simulation. Moreover, MAST updates statistics of each player proportionally to the payoff that the player obtained in the playout. This paper proposes a modification of PPA that updates after each simulation the weights of all the players proportionally to their payoffs. Assume the payoff of the current simulations, $\boldsymbol{q} = \langle q_1, ...q_{|P|} \rangle$, to have values in the interval $[0, 1]$ (note that in GDL scores have values in $[0, 100]$, thus they need to be re-scaled). Given a state s in the playout and the corresponding visited joint move $\boldsymbol{a}^*$, the weights of each player p will be updated as follows:

$$\begin{aligned} W_{(p,a_p^*)} &= W_{(p,a_p^*)} + \alpha \times q_p \\ W_{(p,a)} &= W_{(p,a)} - \alpha \times q_p \times \frac{e^{W_{(p,a)}}}{\sum_{a' \in A(s,p)} e^{W_{(p,a')}}}, \forall a \in A(s,p) \end{aligned} \tag{5}$$

5.2 N-Grams Playout Policy Adaptation

Subsection 3.4 presented the NST strategy, which has been designed by extending MAST to memorize statistics for N-grams of moves other than for single moves. A similar approach can be used to extend the PPA strategy. This paper proposes the N-gram Playout Policy Adaptation (NPPA) strategy, which is similar to PPA, but memorizes weights also for N-grams of moves.

Considering a simultaneous-move game, NPPA creates N-grams of moves in the same way described in Subsection 3.4 for NST. In NPPA, a move for a player in a state of the playout is selected as in PPA, i.e. with Gibbs sampling. The only difference is that the weight of a move is computed as the average of the weights of the N-grams of length 1 to L. This is analogous to Equation 4, but instead of the expected value Q, the weight W is considered for each N-gram. As NST, also NPPA considers the weights of N-grams of length greater than one only if they have been visited more than V times.

The adaptation of weights after a simulation for NPPA is slightly more complex than for NST. While NST updates statistics only of the N-Grams extracted from the simulation, NPPA updates the weights also for the N-grams that end with all the other legal moves in each visited state. More precisely, at the end of a simulation, NPPA extracts all the N-grams for the player p that won the simulation as shown in Equation 3. Subsequently, for each of the extracted N-grams of player p it updates the weights as follows. Let $\langle a_{(t-l+1)}, ..., a_{t-1}, a_t^* \rangle$ be one of the N-grams of length l extracted from the playout for player p, where a_t^* is the move selected by player p in state s at step t of the simulation. For readability, the player index is omitted from the subscript of the moves in the N-gram, and only the simulation step is indicated. NPPA updates the following weights:

$$\begin{aligned} W_{\langle a_{(t-l+1)}, ..., a_{t-1}, a_t^* \rangle} =& W_{\langle a_{(t-l+1)}, ..., a_{t-1}, a_t^* \rangle} + \alpha \\ W_{\langle a_{(t-l+1)}, ..., a_{t-1}, a \rangle} =& W_{\langle a_{(t-l+1)}, ..., a_{t-1}, a \rangle} \\ & - \alpha \times \frac{e^{W_{\langle a_{(t-l+1)}, ..., a_{t-1}, a \rangle}}}{\sum_{a' \in A(s,p)} e^{W_{\langle a_{(t-l+1)}, ..., a_{t-1}, a' \rangle}}}, \forall a \in A(s,p) \end{aligned} \tag{6}$$

6 Empirical Evaluation

This section presents an empirical evaluation of the PPA playout strategy and its enhancements, comparing them with other playout strategies: random, MAST and NST. Multiple series of experiments have been performed. First, Subsection 6.1 describes the experimental setup. Next, Subsections 6.2, 6.3, and 6.4 present the evaluation of the basic version of PPA, the payoff-based adaptation of the policy and the use of N-grams, respectively. Finally, Subsection 6.5 evaluates the performance of MAST, NST and the best version of PPA and NPPA for different simulation budgets, and Subsection 6.6 compares their search time.

6.1 Setup

The strategies evaluated in this paper have been implemented in the framework provided by the open source GGP-Base project [20], as part of an MCTS agent that follows the setup described by Sironi [21]. Experiments are carried out on a set of 14 heterogeneous games taken from the GGP Base repository [19]: 3D Tic Tac Toe, Breakthrough, Knightthrough, Chinook, Chinese Checkers with 3 players, Checkers, Connect 5, Quad (the version played on a 7 × 7 board), Sheep and Wolf, Tic-Tac-Chess-Checkers-Four (TTCC4) with 2 and 3 players, Connect 4, Pentago and Reversi.

Five basic playout strategies are used in the experiments: random, MAST, NST, PPA and NPPA. All of them are combined with the UCT selection strategy with $C = 0.7$. The fixed settings of these strategies are summarized below. For PPA and NPPA some settings vary depending on the experiment, thus, their values will be specified in the corresponding subsections. MAST and NST use an ϵ-greedy strategy to select moves with $\epsilon = 0.4$, and decay statistics after each move with a factor of 0.2 (i.e. 20% of the statistics is reused in the subsequent turn). NST uses N-grams up to length $L = 3$, and considers them only if visited at least $V = 7$ times. All the mentioned parameter values are taken from previous literature, where they were shown to perform well [23, 21]. PPA and NPPA use the Gibbs measure to select moves with $k = 1$, keep statistics memorized between moves and do not decay them. These settings are taken from previous publications on PPA [3–5]. The use of an ϵ-greedy strategy and of statistics decay after moves has been investigated, but preliminary results on PPA did not show any improvement in performance. For NPPA, the same values of $L = 3$ and $V = 7$ that are set for NST are used.

For each performed experiment, two strategies at a time are matched against each other. For each game, all possible assignments of strategies to the roles are considered, except the two configurations that assign the same strategy to all roles. All configurations are run the same number of times until at least 500 games have been played. Each match runs with a budget of 1000 simulations per turn, except for the experiments in Subsection 6.5, that test different budgets. Experimental results always report the average win percentage of one of the two involved strategies with a 95% confidence interval. The average win percentage for a strategy on a game is computed looking at the score. The strategy that

Table 1. Win% of different configurations of PPA against a random playout strategy and against MAST with a budget of 1000 simulations.

Game	**vs random playout strategy**			**vs MAST**	
	PPA$_1$	**PPA$_{0.32}$**	**MAST**	**PPA$_1$**	**PPA$_{0.32}$**
3DTicTacToe	78.2(±3.62)	**89.4**(±2.70)	88.8(±2.77)	38.1(±4.26)	**52.7**(±4.37)
Breakthrough	66.0(±4.16)	**66.4**(±4.14)	62.2(±4.25)	62.2(±4.25)	**63.0**(±4.24)
Knightthrough	51.6(±4.38)	53.4(±4.38)	**85.2**(±3.12)	**12.2**(±2.87)	11.0(±2.75)
Chinook	64.0(±3.28)	65.7(±3.26)	**69.3**(±3.30)	45.9(±3.64)	**50.8**(±3.56)
C.Checkers 3P	64.7(±4.18)	71.4(±3.95)	**74.0**(±3.83)	46.6(±4.36)	**52.4**(±4.36)
Checkers	60.0(±4.00)	68.1(±3.78)	**75.3**(±3.44)	35.6(±3.91)	**37.1**(±3.92)
Connect 5	63.8(±4.17)	75.9(±3.69)	**80.7**(±3.27)	33.2(±3.55)	**49.5**(±3.58)
Quad	73.9(±3.85)	81.7(±3.36)	**82.1**(±3.35)	36.7(±4.18)	**45.7**(±4.33)
SheepAndWolf	59.2(±4.31)	**61.4**(±4.27)	46.0(±4.37)	69.2(±4.05)	**69.4**(±4.04)
TTCC4 2P	75.4(±3.78)	**76.8**(±3.69)	76.0(±3.74)	50.8(±4.37)	**58.1**(±4.32)
TTCC4 3P	59.3(±4.28)	**61.5**(±4.23)	59.2(±4.27)	**53.9**(±4.30)	49.0(±4.34)
Connect 4	24.8(±3.71)	**36.3**(±4.12)	27.3(±3.80)	48.2(±4.30)	**55.9**(±4.27)
Pentago	55.0(±4.32)	62.9(±4.20)	**72.1**(±3.89)	27.1(±3.86)	**39.2**(±4.23)
Reversi	55.3(±4.30)	62.7(±4.19)	**82.8**(±3.26)	21.9(±3.58)	**26.0**(±3.72)
Avg Win%	60.8(±1.12)	66.7(±1.08)	**70.1**(±1.05)	41.6(±1.12)	**47.1**(±1.13)

achieves the highest score is the winner and gets 1 point, if both achieve the highest score they receive 0.5 points each, and it is considered a tie. In each table, bold results represent the highest win percentage for each game in the considered part of the table.

6.2 Basic PPA

This series of experiments evaluates the basic version of the PPA playout strategy presented in Algorithm 1. PPA is compared with MAST by matching both of them against a random playout strategy, and by matching them directly against each other. Two values for the learning rate α are evaluated, 1 and 0.32. The first one is the value used by the first proposed version of PPA [3]. However, subsequent research has shown the value 0.32 to perform better for the PPAF variant of the algorithm [5], therefore it is considered here as well.

Table 1 reports the results, indicating the two PPA variants with PPA$_1$ and PPA$_{0.32}$, respectively. First of all, we can see that both PPA variants and MAST are better than the random playout strategy. Only in Connect 4 they all perform significantly worse than random playouts. When comparing PPA$_1$ and PPA$_{0.32}$, it is visible that the latter has a significantly better overall performance than the former, both against the random playout strategy and against MAST. Moreover, looking at individual games, the performance of PPA$_{0.32}$ is never significantly inferior to the performance of PPA$_1$, being significantly higher in a few of the games. In 3D Tic Tac Toe, Connect 5 and Quad the win rate of PPA$_{0.32}$ is higher both when the opponent uses random playouts and when it uses MAST. Moreover, against the random opponent PPA$_{0.32}$ seems better than PPA$_1$ in Checkers and Connect 4, while against MAST it seems better in Pentago.

When comparing the performance of PPA with the one of MAST, it is clear that the latter has a significantly higher overall win rate. However, for PPA$_{0.32}$ the difference with MAST is smaller, especially when PPA is matched against MAST directly. It is also interesting to look at the performance of MAST and PPA on individual games. MAST seems a particularly suitable strategy for

Table 2. Win% of PPA without and with payoff-based update against MAST with a budget of 1000 simulations and $\alpha = 0.32$.

Game	$\mathbf{PPA_{0.32}}$	$\mathbf{PPA_{\mathit{P}0.32}}$
3DTicTacToe	52.7(±4.37)	**61.0**(±4.26)
Breakthrough	63.0(±4.24)	**65.2**(±4.18)
Knightthrough	11.0(±2.75)	**12.2**(±2.87)
Chinook	50.8(±3.56)	**53.7**(±3.61)
C.Checkers 3P	**52.4**(±4.36)	41.5(±4.31)
Checkers	37.1(±3.92)	**42.6**(±3.99)
Connect 5	49.5(±3.58)	**57.3**(±3.81)
Quad	45.7(±4.33)	**47.2**(±4.34)
SheepAndWolf	69.4(±4.04)	**71.2**(±3.97)
TTCC4 2P	**58.1**(±4.32)	56.0(±4.33)
TTCC4 3P	49.0(±4.34)	**53.1**(±4.33)
Connect 4	**55.9**(±4.27)	51.3(±4.31)
Pentago	39.2(±4.23)	**41.3**(±4.30)
Reversi	26.0(±3.72)	**29.7**(±3.93)
Avg Win%	47.1(±1.13)	**48.8**(±1.14)

Knightthrough, for which it achieves a much higher win rate than PPA. Also for Pentago, Reversi and Checkers MAST seems to perform better than PPA. On the contrary, PPA seems to be particularly suitable for Sheep and Wolf and, when matched against MAST directly, for Breakthrough as well. Finally, when $\alpha = 0.32$, PPA performs significantly better against MAST in TTCC4 with 2 players and Connect 4.

The results presented in Table 1 for Breakthrough and Knightthrough can also be compared with results obtained for these games in previous work on PPA by Cazenave [4]. First of all, as shown by previous literature, Table 1 confirms that PPA and MAST perform significantly better than random playouts on these two games. Moreover, for Knightthrough, MAST clearly performs better than PPA, as it was also shown in previous work, where MAST achieved a win rate of 78.2%, while PPA, for $\alpha = 0.32$, only reached 70.4%. Note that in Table 1 the difference in performance seems to be even higher. This could be due to the fact that the experiments in this paper have some different settings than the ones used by Cazenave. For example, MAST is using an ϵ-greedy strategy to select moves, which has been shown to perform generally better than the one based on the Gibbs measure that is used by Cazenave. Moreover, the search budget used by Cazenave is higher than the 1000 simulations used in Table 1. For Breakthrough, Table 1 seems to suggest that PPA performs slightly better than MAST, while in previous work it seemed to be the opposite, with MAST reaching a win rate of 63.4% and PPA with $\alpha = 0.32$ only of 59.2. However, the difference between MAST and PPA does not seem to be particularly high in either case. The difference between this and previous work might be due to the difference in the experimental setup.

6.3 Payoff-Based Update

This series of experiments looks at the impact of a modification of PPA inspired by MAST: the update of statistics based on the payoffs obtained by the players during a simulation. Given that $\text{PPA}_{0.32}$ performed best in the previous series of experiments, it is used as a baseline for this series of experiments to extend

Table 3. Win% of different configurations of NPPA against a random playout strategy, against NST, and against PPA$_{P0.32}$ with a budget of 1000 simulations.

Game	**vs random playout strategy**			**vs NST**		**vs PPA$_{P0.32}$**	
	NPPA$_1$	**NPPA$_{P0.32}$**	**NST**	**NPPA$_1$**	**NPPA$_{P0.32}$**	**NPPA$_1$**	**NPPA$_{P0.32}$**
3DTicTacToe	71.6(±3.96)	51.4(±4.39)	**93.4**(±2.18)	**20.6**(±3.55)	8.4(±2.43)	**24.8**(±3.79)	9.4(±2.56)
Breakthrough	**73.6**(±3.87)	52.6(±4.38)	68.8(±4.07)	**58.4**(±4.32)	37.0(±4.24)	**57.6**(±4.34)	38.4(±4.27)
Knightthrough	74.0(±3.85)	57.2(±4.34)	**89.8**(±2.66)	**28.4**(±3.96)	17.6(±3.34)	**76.4**(±3.73)	52.0(±4.38)
Chinook	62.2(±3.31)	53.2(±3.50)	**67.9**(±3.45)	**47.8**(±3.70)	33.4(±3.36)	**47.2**(±3.40)	36.3(±3.41)
C.Checkers 3P	**71.4**(±3.95)	53.2(±4.36)	68.5(±4.06)	**53.0**(±4.36)	30.8(±4.03)	**61.9**(±4.24)	35.7(±4.19)
Checkers	72.3(±3.66)	48.4(±4.13)	**73.8**(±3.54)	**48.6**(±4.04)	24.7(±3.44)	**58.8**(±3.92)	34.8(±3.81)
Connect 5	61.6(±4.20)	51.2(±4.39)	**84.0**(±2.99)	**30.5**(±3.47)	18.2(±3.10)	**36.5**(±3.97)	21.8(±3.58)
Quad	75.7(±3.76)	51.8(±4.38)	**87.7**(±2.85)	**38.3**(±4.23)	16.6(±3.25)	**42.8**(±4.31)	19.9(±3.50)
SheepAndWolf	**65.2**(±4.18)	50.6(±4.39)	49.0(±4.39)	**72.2**(±3.93)	52.0(±4.38)	**48.2**(±4.38)	39.0(±4.28)
TTCC4 2P	71.2(±3.95)	53.8(±4.36)	**72.1**(±3.92)	**46.0**(±4.37)	25.7(±3.83)	**45.2**(±4.35)	25.7(±3.83)
TTCC4 3P	**63.5**(±4.16)	52.2(±4.36)	59.8(±4.26)	**55.3**(±4.31)	41.2(±4.25)	**50.7**(±4.35)	39.3(±4.23)
Connect 4	44.5(±4.29)	**48.7**(±4.27)	47.2(±4.28)	42.6(±4.20)	**51.8**(±4.27)	61.1(±4.21)	**67.7**(±3.99)
Pentago	59.6(±4.26)	54.6(±4.33)	**70.2**(±3.96)	**42.7**(±4.29)	30.5(±3.98)	**46.5**(±4.35)	40.5(±4.25)
Reversi	**84.7**(±3.09)	55.5(±4.30)	80.4(±3.40)	**55.3**(±4.31)	18.4(±3.32)	**72.5**(±3.84)	40.6(±4.25)
Avg Win%	67.9(±1.07)	52.5(±1.14)	**72.3**(±1.02)	**45.7**(±1.13)	29.0(±1.03)	**52.2**(±1.14)	35.8(±1.10)

upon. The PPA variant that updates the weights of all players proportionally to their payoff is identified as PPA$_{P0.32}$. Both variants of PPA are matched only against MAST, because it is a more competitive agent than the one that uses random playouts, and it is more interesting to compare against.

Table 2 reports the win rate of the two variants of PPA. Looking at the overall win rate, the payoff-based update of the policy does not seem to have a significant overall impact with respect to the update based only on the winner. Moreover, both variants of PPA are performing worse than MAST, although their performance is rather close. More interesting results emerge when looking at individual games. There are still some games for which both variants of PPA perform significantly better than MAST, namely Breakthrough, Sheep and Wolf and TTCC4 with 2 players. Moreover, PPA$_{P0.32}$ is the variant that performs better than MAST in more games. Other than in the already mentioned games, it is significantly better than MAST also in 3D Tic Tac Toe, Connect 5 and Chinook. For this reason, this is the variant that will be further enhanced and evaluated in the next series of experiments. In addition, this variant might have the advantage of speeding up the learning process because after each simulation it adapts the policy of each player, and not only of the winner.

6.4 N-Grams Playout Policy Adaptation

This series of experiments evaluates the NPPA playout strategy, which behaves like PPA, but in addition uses weights for N-grams of moves. Two settings for the strategy have been tested. The first one corresponds to the initial setting of PPA evaluated in Subsection 6.2 and taken from previous literature [3, 5]. Thus, NPPA with move selection based on Gibbs, policy update only for the winner and $\alpha = 1$ (NPPA$_1$). The second setting corresponds to the best settings found in previous experiments for PPA. Thus, with move selection based on Gibbs, payoff-based policy update for all players, and $\alpha = 0.32$ (NPPA$_{P0.32}$). These two variants of NPPA are compared with NST by matching them all against random playouts and by matching them against each other. Moreover, NPPA is also matched directly against the best version of PPA tested so far, i.e. PPA$_{P0.32}$.

Looking at results in Table 3, it is evident that both NPPA and NST perform overall significantly better then random playouts. In addition, none of them performs significantly worse than random playouts on any of the games, except for NPPA_1 in Connect 4. This seems to confirm results in Table 1, where random playouts seemed to work better than more informed playouts for this game. Comparing NPPA with NST, both variants of the former are performing significantly worse than the latter, especially $\text{NPPA}_{P0.32}$. One of the reasons for this could be that NPPA is updating statistics of N-grams more often than NST. NPPA does not update only statistics of visited N-grams of moves, but also of N-grams that end with all the other legal moves in each visited state. This means that at the start of the search, statistics are updated more often using results of less reliable playouts, and it might take longer for NPPA to converge to optimal values for the policy. This would also explain why $\alpha = 1$ seems to have a better performance than $\alpha = 0.32$. When updating weights, a higher learning rate enables NPPA to give more importance to the result of new and possibly more reliable simulations. There are still a few games where NPPA_1 performs significantly better than NST, namely Breakthrough, TTCC4 with 3 players, Reversi and Sheep and Wolf. In the latter, also $\text{NPPA}_{P0.32}$ shows a good performance.

Results also show that NPPA_1 is significantly better than $\text{NPPA}_{P0.32}$ in almost all the games, and for each of the tree different opponents they are matched against (i.e. random, NST, $\text{PPA}_{P0.32}$). Therefore, the setting that achieved the highest results for PPA do not seem to work as well for NPPA. When matched against $\text{PPA}_{P0.32}$, $\text{NPPA}_{P0.32}$ has a significantly lower win rate in all games except Connect 4. On the contrary, NPPA_1 has a higher overall win rate, and performs significantly worse only on four games (3D Tic Tac Toe, Connect 5, Quad and TTCC4 with 2 players). In Knightthrough, NPPA_1 achieves a much higher win rate than $\text{PPA}_{P0.32}$. This is in part likely due to the settings of $\text{PPA}_{P0.32}$ being sub-optimal for this game, but might also indicate that collecting statistics for N-grams of moves improves the selection of moves during the playout for this game. Finally, the performance of NPPA can be compared with the one of PPA against random playouts, that is reported in Table 2. Considering the best version of NPPA (i.e. NPPA_1), it is visible that its overall performance against random playouts is significantly higher than the one of PPA_1, and at least equal to the one of $\text{PPA}_{0.32}$. Moreover, it is significantly better than $\text{PPA}_{0.32}$ in Knightthrough and Reversi, for which the general PPA strategy does not seem to perform well against MAST.

6.5 Simulation Budget Analysis

This series of experiments analyzes the performance of MAST, $\text{PPA}_{P0.32}$, NST and NPPA_1 with different numbers of simulations per turn. The versions of PPA and NPPA considered for the experiments are the ones that seemed to perform best in previous experiments. PPA and NPPA are updating the statistics of more moves per simulation than MAST and NST, respectively. This means that it might require more time for this statistics to be reliable. This motivates the interest in analyzing how these strategies are influenced by the search budget.

Figure 1 shows how the win rate over all the games of the considered strategies changes against the random playout strategy depending on the budget. For

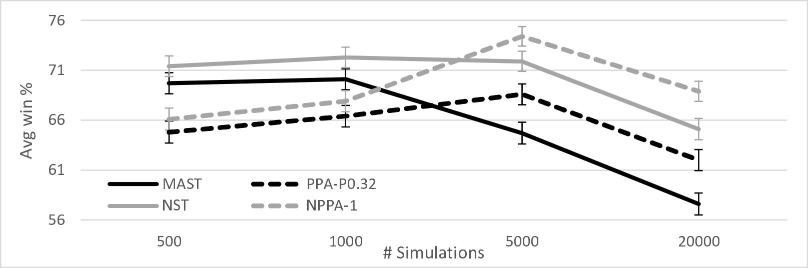

Fig. 1. Win% of MAST, PPA$_{P0.32}$, NST and NPPA$_1$ against a random playout strategy for different numbers of simulations per turn.

most of the considered games, the individual plots look very similar. The plot shows that the performance of all algorithms eventually decreases with more simulations. Moreover, we can see how higher budgets benefit PPA$_{P0.32}$ and NPPA$_1$ more than MAST and NST. According to the law of diminishing returns [14], the performance gain of a search algorithm decreases with the increase of search effort. In this case it looks like the decrease in gain for random playouts is the slowest, thus we notice a decrease of performance over time for all other strategies. However, for PPA$_{P0.32}$ and NPPA$_1$, the decrease in gain seems slower than for NST and MAST. Thus, PPA$_{P0.32}$ and NPPA$_1$ are able to surpass the performance of MAST and NST, respectively, when the budget is increased to 5000 simulations and higher. It seems that PPA$_{P0.32}$ and NPPA$_1$ require a higher number of simulations than MAST and NST in order to converge to a good policy, but then they can surpass the performance of MAST and NST, respectively.

The plot also confirms that the use of N-grams can significantly improve the performance of both MAST and PPA. It also shows that the impact of N-grams on the search is larger for higher budgets. This is not surprising, as it takes some time for the algorithms to collect enough samples for the N-gram statistics to be reliable and positively influence the search.

6.6 Time Analysis

An important aspect to consider when comparing MAST, PPA, NST and NPPA is the computational cost of these strategies. As opposed to random playouts, they invest part of the computational time into memorizing statistics and computing values on which their move-selection is based. Table 4 shows the time in milliseconds that each strategy requires to perform 1000 simulations per turn. These values have been obtained averaging the median time per turn over 500 runs of each game. For consistency, these experiments have been all performed on the same machine, a Linux server consisting of 64 AMD Opteron 6274 2.2-GHz cores. Although all strategies spend time computing statistics, it is immediately clear that both MAST and NST are able to make up for this loss. Both of them never take more time than random playouts to perform 1000 simulations. This indicates that, by biasing the search towards promising moves, they are probably visiting shorter paths that lead to a faster conclusion of the simulated game.

Table 4. Time (ms) that each playout strategy takes to perform 1000 simulations.

Game	**Random**	**MAST**	**PPA$_{P0.32}$**	**NST**	**NPPA$_1$**
3DTicTacToe	352	196	1357	268	2362
Breakthrough	683	226	604	275	1374
Knightthrough	492	166	470	190	633
Chinook	485	281	1568	324	1882
ChineseCheckers3	191	171	539	191	731
Checkers	1974	1923	3545	1661	4164
Connect 5	822	404	2061	520	3100
Quad	435	300	1248	381	1691
SheepAndWolf	471	379	718	406	1008
TTCC4 2P	1069	651	773	745	1465
TTCC4 3P	787	599	650	619	981
Connect 4	102	89	245	98	357
Pentago	394	290	1212	357	1668
Reversi	3562	3493	3521	3433	4525

Table 5. Average win rate over all games of each playout strategy against the random playout strategy, with time budgets of 1 and 5 seconds.

	MAST	**PPA$_{P0.32}$**	**NST**	**NPPA$_1$**
1s	70.1(±1.04)	50.9(±1.14)	73.7(±0.99)	48.0(±1.14)
5s	60.4(±1.09)	52.8(±1.12)	67.3(±1.05)	53.8(±1.12)

Unfortunately, the same effect is not observed for PPA$_{P0.32}$ and NPPA$_1$, except possibly for PPA$_{P0.32}$ in Breakthrough, Knightthrough and TTCC4 with 2 and 3 players, where the search time is lower than the one of random playouts. It could be that PPA$_{P0.32}$ is visiting shorter playouts as well, but the time saved is not sufficient to make up for the extra computation involved. These results are not surprising, given the fact that PPA$_{P0.32}$ and NPPA$_1$ are updating more statistics per simulation than MAST and NST, and indicate a possible limitation of the two former strategies. Although more accurate in the long run, PPA$_{P0.32}$ and NPPA$_1$ might be less suitable than MAST and NST for domains with short time settings, as they might not be able to reach a sufficient number of simulations for their statistics to become accurate and make a positive difference in performance with respect to MAST and NST, respectively. This seems to be confirmed by how the performance of all the strategies changes when they are compared for different time budgets instead of simulation budgets. Table 5 shows the average win rate of the strategies against random playouts over all the considered games. Time budgets of 1 and 5 seconds are considered. Results seem to indicate that the advantage of MAST and NST over random playouts decreases with a higher time budget, while for PPA$_{P0.32}$ and NPPA$_1$ it seems to increase. This means that with more time PPA$_{P0.32}$ and NPPA$_1$ might have an advantage over MAST and NST, respectively. For a couple of games, this already seems to happen with a time budget of 5 seconds. For example, in TTCC4 with 3 players, the win rate of MAST against random playouts changes from 58.7(±4.25) for a budget of 1 second to 56.2(±4.25) for a budget of 5 seconds, while for PPA$_{P0.32}$ it increases from 54.8(±4.28) to 66.0(±4.02). Moreover, the win rate of NST changes from 60.4(±4.20) to 58.8(±4.17), while for NPPA$_1$ it increases from 57.7(±4.28) to 72.6(±3.78). Another example is Sheep and Wolf, for which PPA$_{P0.32}$ performs better than MAST against random playouts both for 1 and 5 seconds. Moreover, for 5 seconds the performance difference between the two strategies seems to become higher. For 1 second, MAST has a win rate of 45.2(±4.37) and PPA$_{P0.32}$ has a win rate of 55.4(±4.36), while for 5 seconds MAST has a win rate of 42.4(±4.34) and PPA$_{P0.32}$ has a win rate of 61.2(±4.28).

7 Conclusion and Future Work

This paper investigated the Playout Policy Adaptation (PPA) strategy as a domain-independent playout strategy for MCTS that can be applied to GGP. First, its formulation for simultaneous-move games has been presented. Next, two enhancements for this strategy have been introduced and evaluated: the update of move statistics of all players proportionally to their payoffs and the use of statistics collected for N-grams of moves (which defines the new NPPA strategy). Both enhancements had already been successfully used for the related MAST playout strategy.

First of all, experiments show that PPA and NPPA are suitable to be applied to a wide variety of games, as they achieve competitive results not only in two-player, sequential-move games, but also in multi-player games (e.g. Chinese Checkers and TTCC4 with 3 players) and simultaneous-move games (e.g. Chinook). It may be concluded that both PPA and NPPA with appropriate settings perform significantly better than random playouts.

Comparing PPA to MAST, both the enhancements that have been shown to work well for MAST seem to also be suitable for PPA. Thus, it may be concluded that using a payoff-based update of the policy or collecting statistics for N-grams is generally not detrimental and seems to improve the performance in a few games. For N-grams, however, it is important to note that the improvement of the performance might depend also on how other parameters are set. The setup of NPPA that performed best is, indeed, different than the best set-up found for PPA. A difference between PPA and MAST is that the former performs better with a higher budget. This is likely because PPA needs more time to collect accurate statistics, but then it might be able to converge to a better policy. The same can be seen when comparing NPPA and NST. Thus, it may be concluded that PPA and NPPA might be more suitable than MAST and NST when a higher simulation budget is available. At the same time, this paper has shown that collecting statistics for PPA and NPPA is generally more time consuming than for MAST and NST. Therefore, the latter might be preferable for domains with limited time settings, like usually happens in GGP competitions.

It should be kept in mind that this work did not explore all combinations of values for the settings of the strategies. Therefore, some good configurations might have been missed. For example, for PPA and NPPA, the α constant could be better tuned. For NPPA the parameters L and V were set to values that performed well for NST, but different values might be better. In addition, other enhancements for the PPA and NPPA strategies could be evaluated. For example, previous research has shown that including information about the length of the simulation when updating the statistics might improve the quality of the search [15, 6]. It is also worth noting that no single strategy and no single setting for a given strategy is able to perform best on all games. This shows that, in a domain like GGP, the search has to be adapted to each game online, instead of using fixed strategies with fixed control-parameters. This could be another promising direction for future work. Finally, it would be important to look into ways to speed up the computation of statistics for PPA and NPPA, so that agents can take advantage of accurate statistics even with limited time settings.

Acknowledgments

This work was supported in part by the French government under management of Agence Nationale de la Recherche as part of the “Investissements d’avenir” program, reference ANR19-P3IA-0001 (PRAIRIE 3IA Institute).

References

1. Auer, P., Cesa-Bianchi, N., Fischer, P.: Finite-time analysis of the multiarmed bandit problem. Machine Learning **47**(2–3), 235–256 (2002)
2. Browne, C.B., Powley, E., Whitehouse, D., Lucas, S.M., Cowling, P.I., Rohlfshagen, P., Tavener, S., Perez, D., Samothrakis, S., Colton, S.: A survey of Monte Carlo tree search methods. IEEE Transactions on Computational Intelligence and AI in Games **4**(1), 1–43 (2012)
3. Cazenave, T.: Playout policy adaptation for games. In: Advances in Computer Games. pp. 20–28. Springer (2015)
4. Cazenave, T.: Playout policy adaptation with move features. Theoretical Computer Science **644**, 43–52 (2016)
5. Cazenave, T., Diemert, E.: Memorizing the playout policy. In: Workshop on Computer Games. pp. 96–107. Springer (2017)
6. Cazenave, T., Saffidine, A., Schofield, M., Thielscher, M.: Nested Monte Carlo search for two-player games. In: Thirtieth AAAI Conference on Artificial Intelligence. pp. 687–693 (2016)
7. Cazenave, T., Teytaud, F.: Application of the nested rollout policy adaptation algorithm to the traveling salesman problem with time windows. In: International Conference on Learning and Intelligent Optimization. pp. 42–54. Springer (2012)
8. Coulom, R.: Efficient selectivity and backup operators in Monte-Carlo tree search. In: van den Herik, H.J., Ciancarini, P., Donkers, H.H.L.M. (eds.) Computers and Games. LNCS, vol. 4630, pp. 72–83. Springer (2007)
9. Edelkamp, S., Gath, M., Cazenave, T., Teytaud, F.: Algorithm and knowledge engineering for the TSPTW problem. In: 2013 IEEE Symposium on Computational Intelligence in Scheduling (CISched). pp. 44–51. IEEE (2013)
10. Edelkamp, S., Gath, M., Greulich, C., Humann, M., Herzog, O., Lawo, M.: Monte-Carlo tree search for logistics. In: Clausen, U., Friedrich, H., Thaller, C., Geiger, C. (eds.) Commercial Transport. pp. 427–440. LNL, Springer (2016)
11. Finnsson, H., Björnsson, Y.: Learning simulation control in General Game-Playing agents. In: Twenty-Fourth AAAI Conference on Artificial Intelligence (AAAI). pp. 954–959. AAAI Press (2010)

12. Genesereth, M., Love, N., Pell, B.: General game playing: Overview of the AAAI competition. AI Magazine **26**(2), 62–72 (2005)
13. Genesereth, M., Thielscher, M.: General Game Playing, Synthesis Lectures on Artificial Intelligence and Machine Learning, vol. 8. Morgan & Claypool Publishers (2014)
14. Heinz, E.A.: Self-play, deep search and diminishing returns. ICGA Journal **24**(2), 75–79 (2001)
15. Kocsis, L., Szepesvári, C.: Bandit based Monte-Carlo planning. In: Fürnkranz, J., Scheffer, T., Spiliopoulou, M. (eds.) Machine Learning: ECML 2006, LNCS, vol. 4212, pp. 282–293. Springer (2006)
16. Love, N., Hinrichs, T., Haley, D., Schkufza, E., Genesereth, M.: General Game Playing: Game Description Language Specification. Tech. rep., Stanford Logic Group (2008)
17. Powley, E.J., Whitehouse, D., Cowling, P.I.: Bandits all the way down: UCB1 as a simulation policy in Monte Carlo tree search. In: 2013 IEEE Conference on Computational Intelligence in Games (CIG). pp. 81–88. IEEE (2013)
18. Rosin, C.D.: Nested rollout policy adaptation for Monte Carlo tree search. In: IJCAI. pp. 649–654 (2011)
19. Schreiber, S.: Games - base repository. http://games.ggp.org/base/ (2016)
20. Schreiber, S., Landau, A.: The General Game Playing base package. https://github.com/ggp-org/ggp-base (2016)
21. Sironi, C.F.: Monte-Carlo Tree Search for Artificial General Intelligence in Games. Ph.D. thesis, Maastricht University (2019)
22. Soemers, D.J.N.J., Sironi, C.F., Schuster, T., Winands, M.H.M.: Enhancements for real-time Monte-Carlo tree search in general video game playing. In: 2016 IEEE Conference on Computational Intelligence and Games (CIG). pp. 436–443. IEEE (2016)
23. Tak, M.J.W., Winands, M.H.M., Björnsson, Y.: N-grams and the Last-Good-Reply policy applied in General Game Playing. IEEE Transactions on Computational Intelligence and AI in Games **4**(2), 73–83 (2012)