Monte Carlo Tree Search in Simultaneous Move Games with Applications to Goofspiel

Marc Lanctot[1], Viliam Lisý[2], and Mark H.M. Winands[1]

[1]Department of Knowledge Engineering, Maastricht University, The Netherlands

[2]Department of Computer Science Czech Technical University in Prague

{marc.lanctot,m.winands}@maastrichtuniversity.nl,lisy@agents.fel.cvut.cz

Abstract. Monte Carlo Tree Search (MCTS) has become a widely popular sampled-based search algorithm for two-player games with perfect information. When actions are chosen simultaneously, players may need to mix between their strategies. In this paper, we discuss the adaptation of MCTS to simultaneous move games. We introduce a new algorithm, Online Outcome Sampling (OOS), that approaches a Nash equilibrium strategy over time. We compare both head-to-head performance and exploitability of several MCTS variants in Goofspiel. We show that regret matching and OOS perform best and that all variants produce less exploitable strategies than UCT.

1 Introduction

Monte Carlo Tree Search (MCTS) is a simulation-based search technique often used in extensive-form games [9, 16]. Having first seen practical success in computer Go [13], MCTS has since been applied successfully to general game playing, real-time and continuous domains, multi-player games, single-player games, imperfect information games, computer games, and more [4].

Despite its empirical success, formal guarantees of convergence of MCTS to the optimal action choice were analyzed only for a MCTS variant called UCT [16], in the case of two-player zero-sum perfect-information sequential (turn-taking) games. In this paper, we focus on MCTS in zero-sum games with perfect information and simultaneous moves. We argue that a good search algorithm for this class of games should converge to a Nash equilibrium (NE) of the game, which is not the case for a variant of UCT [25], commonly used in this setting. Other variants of MCTS, which may converge to NE were suggested [26], but this property was never proven or experimentally evaluated.

In this paper, we introduce Online Outcome Sampling (OOS), a MCTS algorithm derived from Monte Carlo counterfactual regret minimization [17], which provably converges to NE in this class of games. We provide experimental evidence that OOS and several other variants of MCTS, based on Exp3 and Regret matching, also converge to NE in a smaller version of the card game Goofspiel. In addition, we compare the head-to-head performance of five different MCTS variants in full-size Goofspiel. Since Goofspiel has recently been solved [21], we use the optimal minimax values of every state to estimate the exploitability (*i.e.,* worst-case regret) of the strategies used in the full game. The results show that regret matching and an optimized form of OOS (OOS$^+$), which have never been used in context of MCTS, produce the strongest Goofspiel players.

1.1 Related Work

The first application of MCTS to simultaneous move games was in general game playing (GGP) [11] programs. The Cadiaplayer [12] using a strategy we describe as DUCT in Subsection 3.1 was the top performing player of the GGP competition between 2007 and 2009. Despite this success, Shafiei *et al.* [25] provide a counter-example showing that this straightforward application of UCT does not converge to NE even in the simplest simultaneous move games and that a player playing a NE can exploit this strategy. Another variant of UCT, which has been applied to the simultaneous move game Tron [24], builds the tree as if the players were moving sequentially giving one of the player unrealistic informational advantage. This approach also cannot converge to NE in general.

For this reason, other variants of MCTS were considered for simultaneous move games. Teytaud and Flory [26] describe a search algorithm for games with short-term imperfect information, which are a generalization of simultaneous move games. Their algorithm uses Exp3 (see Subsection 3.2) for the simultaneous moves and was shown to work well in the Internet card game Urban Rivals. A more thorough investigation of different selection policies including UCB, UCB1-Tuned, ϵ-greedy, Exp3, and more is reported in the game of Tron [20]. We show a similar head-to-head performance comparison for Goofspiel in Section 4 and we add an analysis of convergence to NE.

Finnsson applied simultaneous move MCTS to several games, including small games of Goofspiel [12, Chapter 6]. This work focused mainly on pruning provably dominated moves. Their algorithm uses solutions to linear programs in the framework of Score-Bounded MCTS [6] to extend the ideas of MCTS-Solver [27] to simultaneous move games. Saffidine *et al.* [23] and Bosansky *et al.* [3] recently described methods for $\alpha\beta$ pruning in simultaneous move games, and also applied their algorithms to simplified Goofspiel. Our work differs in that our algorithm is built with the simulation-based search framework of Monte Carlo Tree Search (MCTS), which is more suitable for larger games with difficult evaluation of the quality of intermediate game states.

The ideas presented in this paper are different than MMCTS and IS-MCTS [2, 10] in the sense that the imperfect information that arises in simultaneous move games is rather short term because it only occurs between state transitions. In our case game trees may include chance events, but the outcomes of the chance events are observable by each player. As a result, techniques such as backward induction [5, 21, 22] are applicable, and search algorithms can be seen as sample-based approximations of these solvers.

2 Simultaneous Move Games

A finite game with simultaneous moves and chance can be described by a tuple $(\mathcal{N}, \mathcal{S} = \mathcal{D} \cup \mathcal{C} \cup \mathcal{Z}, \mathcal{A}, \mathcal{T}, \Delta_c, u_i, s_0)$. The player set $\mathcal{N} = \{1, 2, c\}$ contains player labels, where c denotes the chance player and by convention a player is denoted $i \in \mathcal{N}$. $\mathcal{S}$ is a set of states, with $\mathcal{Z}$ denoting the terminal states, $\mathcal{D}$ the states where players make decisions, and $\mathcal{C}$ the possibly empty set of states where chance events occur. $\mathcal{A} = \mathcal{A}_1 \times \mathcal{A}_2$ is the set of joint actions of individual players. We denote $\mathcal{A}_i(s)$ the actions available to player i in state $s \in \mathcal{S}$. The transition function $\mathcal{T} : \mathcal{S} \times \mathcal{A}_1 \times \mathcal{A}_2 \mapsto \mathcal{S}$ defines the successor state given a current state and actions for both players. $\Delta_c : \mathcal{C} \mapsto \Delta(\mathcal{S})$ describes a probability distribution over possible successor states of the chance event. The utility functions $u_i : \mathcal{Z} \mapsto [v_{\min}, v_{\max}] \subseteq \mathbb{R}$ gives the utility of player i, with v_{min} and $v_{\max}$ denoting the minimum and maximum possible utility respectively. We assume constant-sum games: $\forall z \in \mathcal{Z}, u_1(z) = k - u_2(z)$. The game begins in an initial state s_0.

A *matrix game* is a single step simultaneous move game with action sets $\mathcal{A}_1$ and $\mathcal{A}_2$. Each entry in the matrix A_{rc} where $(r, c) \in A_1 \times A_2$ corresponds to a payoff (to player 1) if row r is chosen by player 1 and column c by player 2. For example, in Matching Pennies, each player has two actions (heads or tails). The row player receives a payoff of 1 if both players choose the same action and 0 if they do not match. Two-player simultaneous move games are sometimes called *stacked matrix games* because at every state s there is a joint action set $\mathcal{A}_1(s) \times \mathcal{A}_2(s)$ that either leads to a terminal state or (possibly after a chance transition) to a subgame which is itself another stacked matrix game.

A *behavioral strategy* for player i is a mapping from states $s \in \mathcal{S}$ to a probability distribution over the actions $\mathcal{A}_i(s)$, denoted $\sigma_i(s)$. Given a profile $\sigma = (\sigma_1, \sigma_2)$, define the probability of reaching a terminal state z under σ as $\pi^\sigma(z) = \pi_1(z)\pi_2(z)\pi_c(z)$, where each $\pi_i(z)$ is a product of probabilities of the

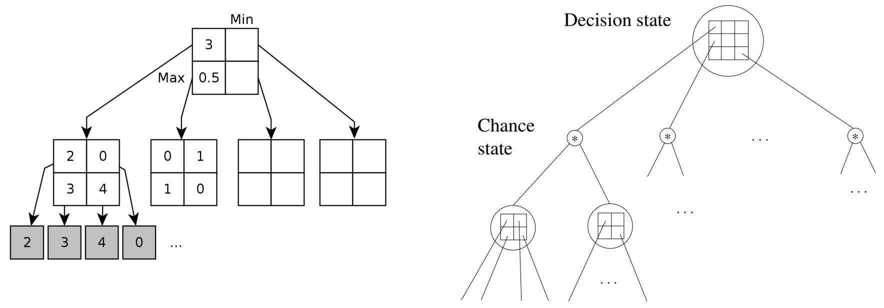

Fig. 1: Examples of a two-player simultaneous game without chance nodes (left) which has Matching Pennies as a subgame, and a portion of 3-card Goofspiel including chance nodes (right). The dark squares are terminal states. The values shown are optimal values that could be obtained by backward induction.
Note: the left figure is taken from [3] and provided by Branislav Bosansky.

actions taken by player i along the path to z (c being chance's probabilities). Define Σ_i to be the set of behavioral strategies for player i. A Nash equilibrium profile in this case is a pair of behavioral strategies optimizing

$$V^* = \max_{\sigma_1 \in \Sigma_1} \min_{\sigma_2 \in \Sigma_2} \mathbb{E}_{z \sim \sigma}[u_1(z)] = \max_{\sigma_1 \in \Sigma_1} \min_{\sigma_2 \in \Sigma_2} \sum_{z \in Z} \pi^{\sigma}(z) u_1(z). \quad (1)$$

In other words, none of the players can improve their utility by deviating unilaterally. For example, the Matching Pennies matrix game has a single state and the only equilibrium strategy is to mix equally between both actions, *i.e.,* play with a *mixed strategy* (distribution) of $(0.5, 0.5)$ giving an expected payoff of $V^* = 0.5$. If the strategies also optimize Equation 1 in each subgame starting in an arbitrary state, the equilibrium strategy is termed subgame perfect.

In two-player constant sum games a (subgame perfect) Nash equilibrium strategy is often considered to be optimal. It guarantees the payoff of at least V^* against any opponent. Any non-equilibrium strategy has its nemesis, which will make it win less than V^* in expectation. Moreover, subgame perfect NE strategy can earn more than V^* against weak opponents. After the opponent makes a sub-optimal move, the strategy will never allow it to gain the loss back. The value V^* is known as the minimax-optimal value of the game and is the same for every equilibrium profile by von Neumann's minimax theorem.

A two-player simultaneous move game is a specific type of two-player imperfect information extensive-form game. In imperfect information games, states are grouped into *information sets*: two states $s, s' \in I$ if the player to act at I cannot distinguish which of these states the game is currently in. Any simultaneous move game can be modeled using an information set to represent a half-completed transition, *i.e.,* $\mathcal{T}(s, a_1, ?)$ or $\mathcal{T}(s, ?, a_2)$.

The model described above is similar to a two-player finite horizon Markov Game [19] with chance events. Examples of such games are depicted in Figure 1.

3 Simultaneous Move Monte Carlo Tree Search

Monte Carlo Tree Search (MCTS) [9, 16] is a simulation-based search algorithm often used in game trees. The main idea is to iteratively run simulations to a terminal state, incrementally growing a tree rooted at the current state. In its simplest form, the tree is initially empty and a single leaf is added each iteration. The nodes in the tree represent game states (decision nodes) or chance events (chance nodes). Each simulation starts by visiting nodes in the tree, selecting (or sampling) which actions to take based on information maintained in the node, and then consequently transitioning to the successor states. When a node is visited whose immediate children are not all in the tree, the node is expanded by adding a new leaf to the tree. Then, a rollout policy is applied from the new leaf to a terminal state. The outcome of the simulation is then back-propagated to all the nodes that were visited during the simulation.

In Simultaneous Move MCTS (SM-MCTS), the main difference is that a joint action is selected. The convergence to an optimal strategy depends critically on

```
1  SM-MCTS(node $s$)
2      if $s$ is a terminal state $(s \in \mathcal{Z})$ then return $u_1(s)$
3      else if $s \in T$ and $s$ is a chance node $(s \in \mathcal{C})$ then
4          Sample $s' \sim \Delta_c(s)$
5          if $s' \notin T$ then add $s'$ to $T$
6          $u_1 \leftarrow$ SM-MCTS($s'$)
7          $X_s \leftarrow X_s + u_1;\ n_s \leftarrow n_s + 1$
8          return $u_1$
9      else if $s \in T$ and $\exists(a_1, a_2) \in \mathcal{A}_1(s) \times \mathcal{A}_2(s)$ not previously selected then
10         Choose one of the previously unselected $(a_1, a_2)$ and $s' \leftarrow \mathcal{T}(s, a_1, a_2)$
11         Add $s'$ to $T$
12         $u_1 \leftarrow$ Rollout($s'$)
13         $X_{s'} \leftarrow X_{s'} + u_1;\ n_{s'} \leftarrow n_{s'} + 1$
14         Update($s, a_1, a_2, u_1$)
15         return $u_1$
16     $(a_1, a_2) \leftarrow$ Select($s$)
17     $s' \leftarrow \mathcal{T}(s, a_1, a_2)$
18     $u_1 \leftarrow$ SM-MCTS($s'$)
19     Update($s, a_1, a_2, u_1$)
20     return $u_1$
```

Algorithm 1: Simultaneous Move Monte Carlo Tree Search

the selection and update policies applied, which are not as straightforward as in purely sequential games. Algorithm 1 describes a single simulation of SM-MCTS. T represents the MCTS tree in which each state is represented by one node. Every node s maintains a cumulative reward sum over all simulations through it, X_s, and a visit count n_s, both initially set to 0. As with standard MCTS, when a state is visited these values are incremented, in the same way on lines 7 and 13, and in the node updates on lines 14 and 19. As seen in Figure 1, a matrix of references to the children is maintained at each decision node.

Chance nodes are explicitly added to the tree and handled between lines 3 and 7, which is skipped in games without chance events since $|\mathcal{C}| = 0$. At a chance node s, $\bar{X}_s = X_s/n_s$ represents the mean value of the chance node and corresponding joint action at the parent of s. This mean value at chance nodes approximates the expected value (weighted sum) that would be computed by backward induction or a depth-limited search algorithm.

At a decision node s, the estimated values $\bar{X}_{s'}$ of the children nodes $s' = \mathcal{T}(s, a_1, a_2)$ over all joint actions form an estimated payoff matrix for node s. The critical parts of the algorithm are the updates on lines 14 and 19 and the selection on line 16. Each variant below will describe a different way to select a joint action and update a decision node.

In practice, there are several optimizations to the base algorithm that might be desirable. For example, if a game has a large branching factor, it may take many iterations for the expansion condition and consequence in lines 9 to 10 to fill up the matrix before switching to a selection policy. The matrix can instead be filled such that at least one action has been taken from each row and one from each column before switching to the selection policy. Since DUCT and Exp3 do not require values for each entry in the matrix, this could reduce the number of simulations before switching to $|\mathcal{A}_1(s)| + |\mathcal{A}_2(s)|$ from $|\mathcal{A}_1(s)||\mathcal{A}_2(s)|$. The use of progressive widening [7, 8] may also lead to deeper searches. In this paper, the implementation for experiments is based on the pseudo-code presented in Algorithm 1.

3.1 Decoupled UCT

In Decoupled UCT (DUCT) [11], each player i maintains separate reward sums $X^i_{s,a}$ and visit counts $n^i_{s,a}$ for their own action set $a \in \mathcal{A}_i(s)$. When a joint action needs to be selected on line 16, each player selects an action that maximizes the UCB value over their reward estimates independently:

$$a_i = \operatorname*{argmax}_{a \in A_i(s)} \left\{ \bar{X}^i_{s,a} + C_i \sqrt{\frac{\ln n_s}{n_{s,a}}} \right\}, \text{ where } \bar{X}^i_{s,a} = \frac{X^i_{s,a}}{n_{s,a}} \quad (2)$$

The update policy increases the rewards and visit counts for each player i: $X^i_{s,a_i} \leftarrow X^i_{s,a_i} + u_i$, and $n_{s,a_i} \leftarrow n_{s,a_i} + 1$.

While references to children nodes in the MCTS tree are maintained in a matrix, each player *decouples* the values and estimates from the joint actions space. In other words, for some state s, each player maintains their own tables of values. For example, suppose the actions sets are $\mathcal{A}_1(s) = \{a, b, c\}$ and $\mathcal{A}_2(s) = \{A, B, C\}$, then the information maintained by at state s is depicted in Figure 2. Many of the other selection policies also maintain values separately, and some use jointly maintained values.

Player 1		
Action	Reward Sum	Visit Count
a	$X^1_{s,a}$	$n_{s,a}$
b	$X^1_{s,b}$	$n_{s,b}$
c	$X^1_{s,c}$	$n_{s,c}$

Player 2		
Action	Reward Sum	Visit Count
A	$X^2_{s,A}$	$n_{s,A}$
B	$X^2_{s,B}$	$n_{s,B}$
C	$X^2_{s,C}$	$n_{s,C}$

Fig. 2: Decoupled values maintained in the tree at a node representing state s.

After the simulations, a move is chosen that maximizes $\bar{X}^i_{s,a_i}$ for the searching player i. Alternatively, one can choose to play a mixed (*i.e.,* randomized) strategy by normalizing the visit counts. We call the former DUCT(max) and the latter DUCT(mix).

3.2 Exp3

In Exp3 [1], each player maintains an estimate of the sum of rewards, denoted $\hat{x}^i_{s,a}$, and visit counts $n^i_{s,a}$ for each of their actions. The joint action selected on line 16 is composed of an action independently selected for each player based on the probability distribution. This probability of sampling action a_i is

$$\sigma^t_i(s, a_i) = \frac{(1-\gamma)\exp(\eta w^i_{s,a_i})}{\sum_{a_j \in \mathcal{A}_i(s)} \exp(\eta w^i_{s,a_j})} + \frac{\gamma}{|\mathcal{A}_i(s)|}, \text{ where} \tag{3}$$

$$\eta = \frac{\gamma}{|\mathcal{A}_i(s)|}, \text{ and } w^i_{s,a} = \hat{x}^i_{s,a} - \max_{a' \in \mathcal{A}_i(s)} \hat{x}^i_{s,a'}.$$

Here, the reason to use $w^i_{s,a}$ is for numerical stability in the implementation. The action selected by normalizing over the maximum value will be identical to the action chosen without normalizing.

The update after selecting actions (a_1, a_2) and obtaining a simulation result (u_1, u_2) updates the visits count and adds to the corresponding reward sum estimates the reward divided by the probability that the action was played by the player using

$$n_{s,a_i} \leftarrow n_{s,a_i} + 1, \quad \hat{x}^i_{s,a_i} \leftarrow \hat{x}^i_{s,a_i} + \frac{u_i}{\sigma^t_i(s, a_i)}.$$

Dividing the value by the probability of selecting the corresponding action makes $\hat{x}^i_{s,a}$ estimate the sum of rewards over all iterations, not only the once where a_i was selected.

Since these values and strategies are maintained separately for each player, Exp3 is decoupled in the same sense as DUCT, storing values separately as depicted by Figure 2.

The mixed strategy used by player i after the simulations are done is given by the frequencies of visit counts of the actions,

$$\sigma^{final}_i(s, a_i) = \frac{n_{s,a_i}}{\sum_{b_i \in A_i(s)} n_{s,b_i}}.$$

Previous work [26] suggests first removing the samples caused by the exploration. This modification proved to be useful also in our experiments, so before computing the resulting final mixed strategy, we set

$$n_{s,a_i} \leftarrow \max\left(0, n_{s,a_i} - \frac{\gamma}{|A_i(s)|} \sum_{b_i \in A_i(s)} n_{s,b_i}\right). \tag{4}$$

3.3 Regret Matching

This variant applies regret matching [15] to the current estimated matrix game at each stage. Suppose iterations are numbered from $t \in \{1, 2, 3, \cdots\}$ and at each iteration and each decision node s there is a mixed strategy $\sigma_i^t(s)$ used by each player i for each node s in the tree, initially set to uniform random: $\sigma_i^0(s,a) = 1/|\mathcal{A}(s)|$. Each player i maintains a cumulative regret $r_s^i[a]$ for having played $\sigma_i^t(s)$ instead of $a \in \mathcal{A}_i(s)$. In addition, a table for the average strategy is maintained per player as well $\bar{\sigma}_s^i[a]$. The values in both tables are initially set to 0.

On iteration t, the selection policy (line 16 in Algorithm 1) first builds the player's current strategies from the cumulative regret. Define $x^+ = \max(x, 0)$,

$$\sigma_i^t(s,a) = \frac{r_s^i[a]}{R_{sum}^+} \text{ if } R_{sum}^+ > 0 \text{ oth. } \frac{1}{|\mathcal{A}_i(s)|}, \text{ where } R_{sum}^+ = \sum_{a \in \mathcal{A}_i(s)} r_s^{i,+}[a]. \quad (5)$$

The main idea is to adjust the strategy by assigning higher weight proportionally to actions based on the regret of having not taken them over the long-term. To ensure exploration, an γ-on-policy sampling procedure similar to Equation 3 is used choosing action a with probability $\gamma/|\mathcal{A}(s)| + (1-\gamma)\sigma_i^t(s,a)$.

The updates on line 14 and 19 add regret accumulated at the iteration to the regret tables r_s^i and the average strategy $\bar{\sigma}_s^i[a]$. Suppose joint action (a_1, a_2) is sampled from the selection policy and utility u_i is returned from the recursive call on line 18. Label the current child (i,j) estimate $\bar{X}_{s,i,j}$ and the $reward(i,j) = \bar{X}_{s,i,j}$ if $(i,j) \neq (a_1, a_2)$, or u_i otherwise. The updates to the regret are:

$$\forall a_1' \in \mathcal{A}_1(s), r_s^1[a_1'] \leftarrow r_s^1[a_1'] + (reward(a_1', a_2) - u_1),$$
$$\forall a_2' \in \mathcal{A}_2(s), r_s^2[a_2'] \leftarrow r_s^2[a_2'] + (reward(a_1, a_2') - u_2),$$

and average strategy updates for each player, $\bar{\sigma}_s^i[a] \leftarrow \bar{\sigma}_s^i[a] + \sigma_i^t(s,a)$.

The regret values $r_s^i[a_i]$ are maintained separately by each player, as in DUCT and depicted by Figure 2. However, the updates and specifically the reward uses a value that is a function of the joint action space.

After the simulations, a move for the root s is chosen by sampling over the strategy obtained by normalizing the values in $\bar{\sigma}_s^i$.

3.4 Online Outcome Sampling

Online Outcome Sampling (OOS) is an MCTS adaptation of the outcome sampling MCCFR algorithm designed for offline equilibrium computation in imperfect information games [17]. Regret matching is applied but to a different type of regret, the sampled counterfactual regret. Counterfactual regret is a way to define individual regrets at s for not having played actions $a \in \mathcal{A}_i(s)$ weighted by the probability that the opponent played to reach s [28]. The sampled counterfactual regret is an unbiased estimate of the counterfactual regret.

In OOS, each simulation chooses a single exploration player i_{exp}, which alternates across simulations. Also, the probability of sampling to a state s due to the exploring player's selection policy, π, is maintained. These two parameters are added to the function in line 1 of Algorithm 1. Define $\sigma_i^t(s)$, regret and average strategy tables as in Subsection 3.3. Regret matching (Equation 5) is used to build the strategies, and the action selected for $i = i_{exp}$ is sampled with probability $p_{s,a_i} = \gamma/|\mathcal{A}(s)| + (1-\gamma)\sigma_i^t(s, a_i)$. The other player j's action is selected with probability $p_{s,a_j} = \sigma_j^t(s, a_j)$. The recursive call on line 18 then sends down $\pi p_{s,a_i}$ as the new sample probability.

Upon return from the recursive call, the exploring player $i = i_{exp}$ first builds a table of expected values given their strategies $v_s^i[a]$. In outcome sampling, the values assigned to nodes that were not sampled are assigned a value of 0. This ensures that the estimate of the true counterfactual values remains unbiased. Due to the complexity of the implementation we omit this standard version of outcome sampling and refer interested readers to [18, Chapter 4]. Instead, we present a simpler optimized form inspired by Generalized MCCFR with probing [14] that seems to perform better in practice in our initial investigation. The idea is to set the value of the unsampled actions to their current estimated value. Define the child state $s_{\{a_i,a_j\}} = \mathcal{T}(s, a_i, a_j)$ if $(i, j) = (1, 2)$ or $\mathcal{T}(s, a_j, a_i)$ otherwise. For the exploring player $i = i_{exp}$, for $a \in \mathcal{A}_i(s)$, the values are:

$$v_s^i[a] = \sum_{a' \in \mathcal{A}_j(s)} \sigma_j^t(s, a') X_{s,a'}^j \text{ where } X_{s,a'}^j = \begin{cases} u_i & \text{if } \{a, a'\} \text{ were selected} \\ \frac{X_{s'}}{n_{s'}} & \text{oth., where } s' = s_{\{a,a'\}} \end{cases}$$

The expected value of the current strategy for the exploring player $i = i_{exp}$ is then $v_{s,\sigma}^i = \sum_{i \in \mathcal{A}_i(s)} \sigma_i^t(s, a) v_s^i[a]$. The regrets are updated for $i = i_{exp}$ and average strategy for $j \neq i_{exp}$ as follows. For all $a_i \in \mathcal{A}_i(s)$ and all $a_j \in \mathcal{A}_j(s)$:

$$r_s^i[a_i] \leftarrow r_s^i[a_i] + \frac{1}{\pi}\left(v_s^i[a_i] - v_{s,\sigma}^i\right), \text{ and}$$

$$\bar{\sigma}_s^j[a_j] \leftarrow \bar{\sigma}_s^j[a_j] + \frac{1}{\pi}\sigma_j^t(s, a_j)$$

Finally, after all the simulations a move is chosen for player i by [21] selecting an action from the mixed strategy obtained by normalizing the values in $\bar{\sigma}_{s_{root}}^i$. We refer to this optimized version of OOS as OOS$^+$.

Since OOS is an application of outcome sampling to the subgame defined by the search tree, it converges to an equilibrium as the number of iterations at the same rate as outcome sampling MCCFR [18]. OOS$^+$ introduces bias and hence may not converge to an equilibrium strategy [14]. Approximate observed convergence rates are shown in Subsection 4.3.

By way of example, consider Figure 3. Suppose $i_{exp} = i = 1$, the trajectory sampled is the one depicted giving payoff u_1 to Player 1, and Player 1's sampled

	f	f'
e	1/2	1/5
e'	6/10	7/10

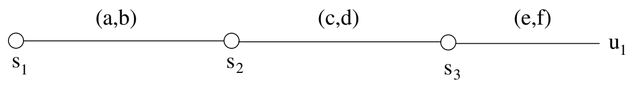

Fig. 3: Example of Online Outcome Sampling.

action sequence is a, c, e. Given this trajectory, Player 1's regret tables and Player 2's average strategies are updated at s_1, s_2, and s_3. Specifically at s_3, the matrix shown contains the reward estimates such that the top-left entry corresponds to $X_{s_3,e,f}/n_{s_3,e,f}$. The probability of sampling s_3 was $\pi = p_{s_1,a} \cdot p_{s_2,c}$. The values $v^i_{s_3}[e] = \sigma_j(s_3, f)u_1 + \sigma_j(s_3, f')/5$, $v^i_{s_3}[e'] = 6\sigma_j(s_3, f)/10 + 7\sigma_j(s_3, f')/10$, and $v^i_{s,\sigma} = \sigma_i(s_3, e)v^i_{s_3}[e] + \sigma_i(s_3, e')v^i_{s_3}[e']$.

4 Empirical Evaluation

In this section we present and discuss the experiments performed to assess the practical behavior of the algorithms above.

4.1 Goofspiel

Goofspiel is a card game where each player gets N cards marked 1-N, and there is a central pile, shuffled and face down called the point-card deck (also 1-N). Every turn, the top card of this point card deck flips, it is called the *upcard*. Then, players choose a *bid* card from their hand and reveal it simultaneously. The player with the higher bid card obtains a number of points equal to the value of the upcard. The bid cards and upcard are then discarded and a new round starts. At the end of N rounds, the player with the highest number of points wins. If the number of points are tied, the game ends in a draw. The standard game of Goofspiel has $N = 13$, which has $(13!)^3 \approx 2.41 \cdot 10^{29}$ unique play sequences including chance events.

There are two ways to define the payoffs received at terminal states. Either the player with the highest points wins (payoffs $\{0, 0.5, 1\}$) or the payoff to the players is the difference in scores. We refer to the former as Win-Loss Goofspiel (WL-Goof(N)) and the latter as Point-Difference Goofspiel (PD-Goof(N)). A backward induction method to solve PD-Goof(N) was originally described in [22] and has recently been implemented and used to solve the game [21] for $N \leq 13$, therefore the optimal minimax value for each state is known. Our evaluation makes use of these in Subsection 4.3. However, WL-Goof(N) is more common in the games and AI community [3, 12, 17, 23].

Mixing between strategies is important in Goofspiel. Suppose a player does not mix and always bids with card n at s. An opponent can respond by playing card $n + 1$ if $n \neq 13$ and $n = 1$ otherwise. This counter-strategy results in collecting every point card except the one lost by the $n = 13$, leading to a victory by a margin of at least 78 points when $N = 13$. This remains true even if the point-card deck was fixed (removing all chance nodes). Nonetheless, the results presented below may differ in a game without chance nodes.

4.2 Head-to-Head Performance

To assess the individual performance of each algorithm, we run a round-robin tournament where each player plays against each other player $n = 10000$ times.

P1 \ P2	RND	DUCT(max)	DUCT(mix)	Exp3	OOS	OOS$^+$	Tuned Parm.
DUCT(max)	76.0						$C_i = 1.5$
DUCT(mix)	78.3	57.5					$C_i = 1.5$
Exp3	80.0	55.8	48.4				$\gamma = 0.2$
OOS	73.1	55.3	43.8	47.0			$\gamma = 0.5$
OOS$^+$	77.7	67.0	53.3	60.0	57.1		$\gamma = 0.55$
RM	80.9	63.3	53.2	57.2	58.3	50.4	$\gamma = 0.025$

P1 \ P2	RND	DUCT(max)	DUCT(mix)	Exp3	OOS	OOS$^+$	Tuned Parm.
DUCT(max)	12.92						$C_i = 150$
DUCT(mix)	11.88	0.91					$C_i = 150$
Exp3	13.18	4.15	3.17				$\gamma = 0.01$
OOS	10.69	3.33	0.82	-1.71			$\gamma = 0.5$
OOS$^+$	10.83	8.08	3.23	1.03	1.03		$\gamma = 0.4$
RM	12.94	6.60	3.41	1.12	1.05	0.17	$\gamma = 0.025$

Table 1: Top: Win percentages for player 1 in WL-Goof(13), 95% confidence interval widths ≤ 1 %. Bottom: Average points earned per game for player 1 in PD-Goof(13). 95% confidence intervals widths ≤ 0.28. 10000 games per matchup. Draws considered half wins to each player to ensure the percentages sum to 100.

This tournament is run using WL-Goof(13) and PD-Goof(13). Parameters are tuned manually by playing against a mix of the players. The metric used to measure performance in WL-Goof is win percentage with 0.5 win for a tie and in PD-Goof is the average number of points gained per game. Each player has 1 second of search time and in our implementation each algorithm generally achieves well above 100000 simulations per second (see Table 2) using a single thread run on a 2.2 GHz AMD Opteron 6174. A uniform random strategy is used for the rollout policy. Ideally we are interested in the performance under different rollout policies, but we leave this as an interesting topic of future work.

The results are shown in Table 1. The RND player chooses a card to play uniformly at random. Of the MCTS variants, we notice that DUCT(max) had the worst performance, losing to every other algorithm in both games. In contrast, RM and OOS had the best performance, winning against every other algorithm in both games. RM's wins and gains against OOS$^+$ are not statistically significant, and OOS$^+$ seems to perform better against the other variants. This may mean that the reach probabilities and counterfactual values are important, even in the simultaneous move setting, the simplest form of imperfect information. However, in both games Exp3 appears to perform better than standard OOS. Also, some results differ between the two games, implying that their relative strength may vary. For example, in WL-Goof, RM wins 58.3% vs. OOS and 53.2% against DUCT(mix) and in PD-Goof wins only 1.05 points vs. OOS compared to 3.41 vs. DUCT(mix).

Algorithm	Mean Ex_2	Mean Ex_4	Mean simulations per second
DUCT(max)	7.43 ± 0.15	12.87 ± 0.13	124127 ± 286
DUCT(mix)	5.10 ± 0.05	7.96 ± 0.02	124227 ± 286
Exp3	5.77 ± 0.10	10.12 ± 0.08	125165 ± 61
OOS	**4.02 ± 0.06**	**7.92 ± 0.04**	186962 ± 361
OOS+	5.59 ± 0.09	9.30 ± 0.08	85940 ± 200
RM	5.56 ± 0.10	9.36 ± 0.07	138284 ± 249

Table 2: Depth-limited exploitability at different depths and relative speeds in PD-Goof(11). 800 search samples per root state, 95% confidence interval widths.

4.3 Exploitability and Convergence

After its simulations, each MCTS algorithm above recommends a play strategy for each state in the tree $\sigma_i(s)$. The exploitability of this strategy can be obtained by computing the amount it can lose against its worst-case opponent. Defined formally, $Ex(s, \sigma_i) = \max_{\sigma_j \in \Sigma_j}(V^*(s) - u_i(s, \sigma_i, \sigma_j))$, where $u_i(s, \sigma_i, \sigma_j)$ is the expected return of the subgame rooted at s when players use (σ_i, σ_j) and $V^*(s)$ is the optimal minimax value of state s. Zero exploitability means that σ_i is a Nash equilibrium strategy. Computing exact exploitability would require a strategy at every state in the game, which may not be well defined after short computation in the root. Therefore, we compute a depth-limited lower bound approximation to this value, which assumes optimal play after depth d:

$$Ex_d(s, \sigma_i) = \begin{cases} V^*(s) & \text{if } d = 0; \\ \sum_{s' \in \Delta_c(s)} \Delta_c(s, s') Ex_{d-1}(s', \sigma_i) & \text{if } s \in \mathcal{C}; \\ \max_{a_j \in \mathcal{A}_j(s)} \sum_{a_i \in \mathcal{A}_i(s)} \sigma_i(s, a_i) Ex_{d-1}(\mathcal{T}(s, a_i, a_j), \sigma_i) & \text{otherwise.} \end{cases}$$

It can be computed using a depth-limited expectimax search.

We assume that the player will not run additional simulations in the following moves and follow the strategy computed in the root until the end of the game. If this strategy is undefined at some point of the game, we assume selecting an arbitrary action. The mean exploitability values for depth $d \in \{2, 4\}$ over every initial upcard in PD-Goof(11), are shown in Table 2.

The results in Table 2 indicate that standard OOS, the only method known to converge to NE, produces the strategies with the lowest depth-limited exploitability for $d \in \{2, 4\}$. However, as seen in Subsection 4.2 this does not necessarily lead to gains in performance, likely due to the restricted search time. Nonetheless, in a repeated play setting where opponents may adapt, less exploitable strategies are desirable. Each of the other algorithms produce less exploitable strategies than DUCT(max), which was expected in Goofspiel due to the importance of mixing. However, surprisingly, DUCT(mix) strategies are much less exploitable than expected. This begs the question of whether DUCT(mix) produces less exploitable strategies in Goofspiel, so in our next experiment we

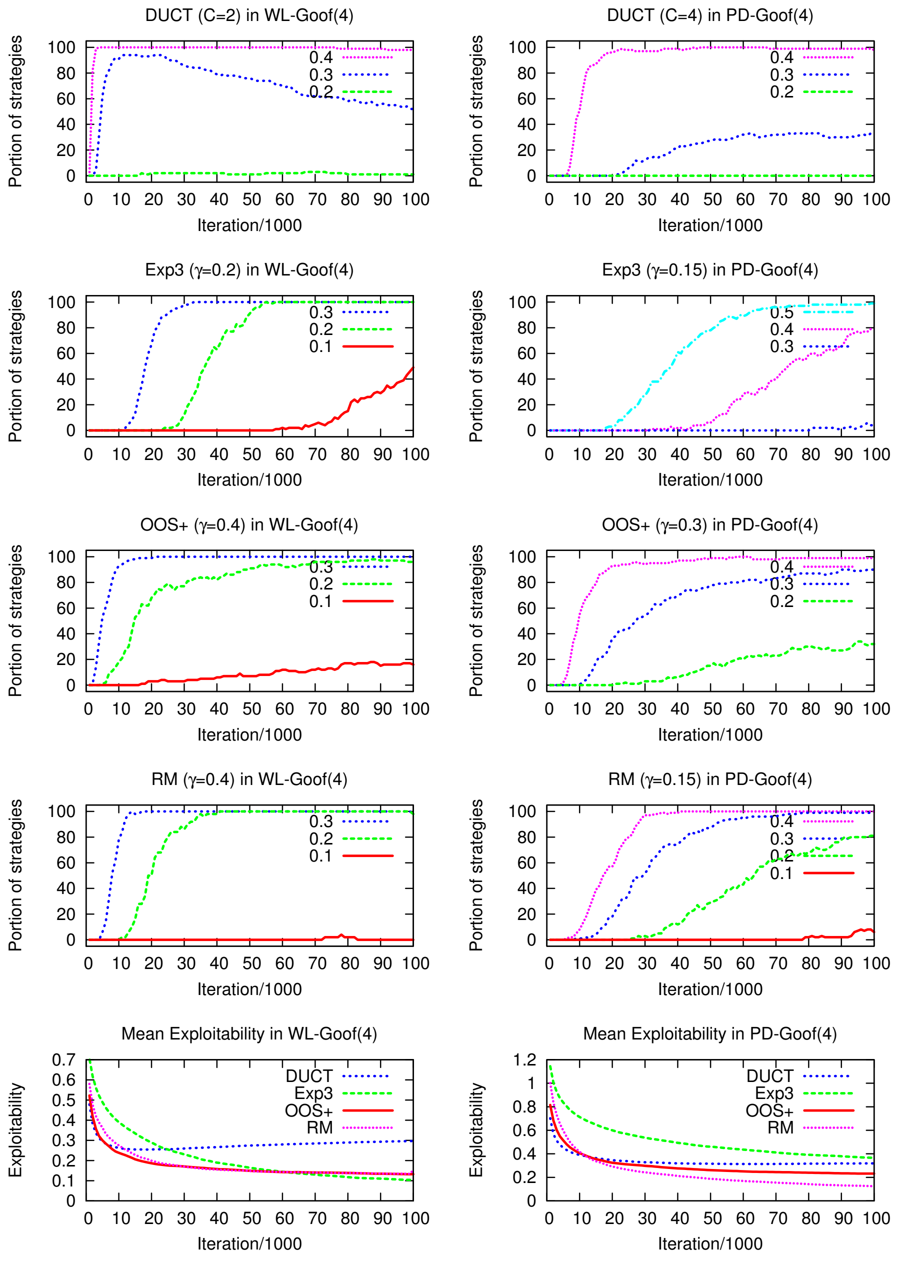

Fig. 4: The percentage of strategies produced by MCTS with exploitability lower than the given threshold after certain number of iterations in WL-Goof(4) (first four in left column), PD-Goof(4) (first four in right column) and mean exploitability for both Goofspiel versions (bottom two).

run the full best response to compute the full-game exploitability in smaller games of Goofspiel. Given the results below, we speculate that DUCT(mix) may be rotating among strategies in the support of an equilibrium strategy recommending a mixed strategy that coincidentally is less exploitable in PD-Goof(11) given the low search time. We do admit that more work is needed to clarify this point.

The next experiment evaluates how quickly the strategy computed by MCTS converges to a Nash equilibrium strategy in smaller game. We run MCTS with each of the selection strategies for 100000 iterations from the root and we computed the value of the full best response against this strategy after every 1000 iterations. The eight graphs in Figure 4 represent the number of runs (out of one hundred), in which the exploitability of the strategy was lower than the given threshold in PD/WL-Goof(4). For example with Exp3 in WL-Goof(4), the exploitability was always smaller than 0.3 after 30 thousand iterations and in 49 out of 100 runs, it was less than 0.1 after 100 thousand iterations. The last two graphs show the mean exploitability of the strategies. Consistently with the previous observations [25], the results show that DUCT does not converge to Nash equilibrium of the game. In fact, the exploitability of the produced strategy starts to increase after 20000 iterations. Exp3, OOS+, and RM strategies converge to the (at least good approximation of) Nash equilibrium strategy in this game. The computed strategies have low exploitability with increasing probability. In WL-Goof(4), OOS+ and RM converge much faster in the earlier iterations, but Exp3 converges more quickly and steadily with more iterations. In PD-Goof(4), RM clearly dominates the other strategies after 20000 iterations.

5 Conclusion and Future Work

In this paper, we compare six different selection strategies for MCTS in games with perfect information and simultaneous moves with respect to actual playing performance in a large game of Goofspiel and convergence to the Nash equilibrium in its smaller variant. The OOS strategy we introduced is the only one, which provably eventually converges to NE. After the whole tree is constructed, the updates behave exactly as in MCCFR, an offline equilibrium computation method with formal guarantees of convergence. The initial finite number of iterations, in which the strategy in some nodes was not updated cannot prevent the convergence. We believe OOS+, RM, and Exp3 also converge to Nash equilibria in this class of games, which we experimentally verify in the small Goofspiel games. We aim to provide the formal proofs and analysis of convergence rates in the future work.

The novel OOS+ and RM strategies have the quickest experimental convergence and performed best also in head-to-head matches. Both have beaten all the other strategies significantly and the performance difference in their mutual matches were insignificant.

In future work, we hope to apply some of these algorithms in the general game-playing and other simultaneous move games, such as Tron and Oshi-Zumo, and compare to existing algorithms such as SMAB and double-oracle methods to better assess their general performance. In addition, we are curious about the effect of different rollout policies on the behavior of each algorithm, the comparison to existing studies in UCT.

Acknowledgments. We would like to thank Laurent Bartholdi for sharing his code for solving Goofspiel. We would also like to thank Olivier Teytaud for advice in optimizing Exp3. This work is partially funded by the Netherlands Organisation for Scientific Research (NWO) in the framework of the project Go4Nature, grant number 612.000.938 and the Czech Science Foundation, grant no. P202/12/2054.

References

1. Auer, P., Cesa-Bianchi, N., Freund, Y., Schapire, R.E.: Gambling in a rigged casino: The adversarial multi-armed bandit problem. In: Proceedings of the 36th Annual Symposium on Foundations of Computer Science. pp. 322–331 (1995)
2. Auger, D.: Multiple tree for partially observable monte-carlo tree search. In: Applications of Eolutionary Computation (EvoApplications 2011), Part I. LNCS, vol. 6624, pp. 53–62. Springer-Verlag, Berlin, Heidelberg (2011)
3. Bosansky, B., Lisy, V., Cermak, J., Vitek, R., Pechoucek, M.: Using double-oracle method and serialized alpha-beta search for pruning in simultaneous moves games. In: Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence (IJCAI). pp. 48–54 (2013)
4. 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)
5. Buro, M.: Solving the Oshi-Zumo game. In: Proceedings of the Advances in Computer Games Conference 10. IFIP Advances in Information and Communication Technology, vol. 135, pp. 361–366 (2003)
6. Cazenave, T., Saffidine, A.: Score bounded Monte-Carlo tree search. In: Proceedings of the 7th International Conference on Computers and Games (CG 2010). LNCS, vol. 6515, pp. 93–104. Springer-Verlag, Berlin, Heidelberg (2011)
7. Chaslot, G.M.J.B., Winands, M.H.M., Uiterwijk, J.W.H.M., van den Herik, H.J., Bouzy, B.: Progressive strategies for monte-carlo tree search. New Mathematics and Natural Computation 4(3), 343–357 (2008)
8. Couetoux, A., Hoock, J.B., Sokolovska, N., Teytaud, O., Bonnard, N.: Continuous upper confidence trees. In: LION'11: Proceedings of the 5th International Conference on Learning and Intelligent Optimization. LNCS, vol. 6683, pp. 433–445 (2011)
9. Coulom, R.: Efficient selectivity and backup operators in Monte-Carlo tree search. In: Proceedings of the 5th International Conference on Computers and Games (CG'06). LNCS, vol. 4630, pp. 72–83. Springer-Verlag, Berlin, Heidelberg (2007)
10. Cowling, P.I., Powley, E.J., Whitehouse, D.: Information set monte carlo tree search. IEEE Transactions on Computational Intelligence and AI in Games 4(2), 120–143 (2012)
11. Finnsson, H.: Cadia-player: A general game playing agent. Master's thesis, Reykjavík University (2007)

12. Finnsson, H.: Simulation-Based General Game Playing. Ph.D. thesis, Reykjavík University (2012)
13. Gelly, S., Kocsis, L., Schoenauer, M., Sebag, M., Silver, D., Szepesvári, C., Teytaud, O.: The grand challenge of computer Go: Monte Carlo tree search and extensions. Communications of the ACM 55(3), 106–113 (2012)
14. Gibson, R., Lanctot, M., Burch, N., Szafron, D., Bowling, M.: Generalized sampling and variance in counterfactual regret minimization. In: Proceedings of the Twenty-Sixth Conference on Artificial Intelligence (AAAI-12). pp. 1355–1361 (2012)
15. Hart, S., Mas-Colell, A.: A simple adaptive procedure leading to correlated equilibrium. Econometrica 68(5), 1127–1150 (2000)
16. Kocsis, L., Szepesvári, C.: Bandit-based Monte Carlo planning. In: 15th European Conference on Machine Learning. LNCS, vol. 4212, pp. 282–293 (2006)
17. Lanctot, M., Waugh, K., Bowling, M., Zinkevich, M.: Sampling for regret minimization in extensive games. In: Advances in Neural Information Processing Systems (NIPS 2009). pp. 1078–1086 (2009)
18. Lanctot, M.: Monte Carlo Sampling and Regret Minimization for Equilibrium Computation and Decision-Making in Large Extensive Form Games. Ph.D. thesis, Department of Computing Science, University of Alberta, Edmonton, Alberta, Canada (2013)
19. Littman, M.L.: Markov games as a framework for multi-agent reinforcement learning. In: In Proceedings of the Eleventh International Conference on Machine Learning. pp. 157–163. Morgan Kaufmann (1994)
20. Perick, P., St-Pierre, D.L., Maes, F., Ernst, D.: Comparison of different selection strategies in monte-carlo tree search for the game of Tron. In: Proceedings of the IEEE Conference on Computational Intelligence and Games (CIG). pp. 242–249 (2012)
21. Rhoads, G.C., Bartholdi, L.: Computer solution to the game of pure strategy. Games 3(4), 150–156 (2012)
22. Ross, S.M.: Goofspiel — the game of pure strategy. Journal of Applied Probability 8(3), 621–625 (1971)
23. Saffidine, A., Finnsson, H., Buro, M.: Alpha-beta pruning for games with simultaneous moves. In: Proceedings of the Thirty-Second Conference on Artificial Intelligence (AAAI-12). pp. 556–562 (2012)
24. Samothrakis, S., Robles, D., Lucas, S.M.: A UCT agent for Tron: Initial investigations. In: Proceedings of the 2010 IEEE Symposium on Computational Intelligence and Games (CIG). pp. 365–371 (2010)
25. Shafiei, M., Sturtevant, N.R., Schaeffer, J.: Comparing UCT versus CFR in simultaneous games. In: Proceeding of the IJCAI Workshop on General Game-Playing (GIGA). pp. 75–82 (2009)
26. Teytaud, O., Flory, S.: Upper confidence trees with short term partial information. In: Applications of Eolutionary Computation (EvoApplications 2011), Part I. LNCS, vol. 6624, pp. 153–162. Springer-Verlag, Berlin, Heidelberg (2011)
27. Winands, M.H.M., Björnsson, Y., Saito, J.T.: Monte-Carlo tree search solver. In: Proceedings of the 6th International Conference on Computers and Games (CG 2008). LNCS, vol. 5131, pp. 25–36. Springer-Verlag, Berlin, Heidelberg (2008)
28. Zinkevich, M., Johanson, M., Bowling, M., Piccione, C.: Regret minimization in games with incomplete information. In: Advances in Neural Information Processing Systems 20 (NIPS 2007). pp. 905–912 (2008)