Monte Carlo Tree Search Variants for Simultaneous Move Games

Mandy J. W. Tak, Marc Lanctot, and Mark H. M. Winands
Games & AI Group, Department of Knowledge Engineering, Maastricht University
{mandy.tak,marc.lanctot,m.winands}@maastrichtuniversity.nl

***Abstract*—Monte Carlo Tree Search (MCTS) is a widely-used technique for game-tree search in sequential turn-based games. The extension to simultaneous move games, where all players choose moves simultaneously each turn, is non-trivial due to the complexity of this class of games. In this paper, we describe simultaneous move MCTS and analyze its application in a set of nine disparate simultaneous move games. We use several possible variants, Decoupled UCT, Sequential UCT, Exp3, and Regret Matching. These variants include both deterministic and stochastic selection strategies and we characterize the game-play performance of each one. The results indicate that the relative performance of each variant depends strongly on the game and the opponent, and that parameter tuning can also not be as straightforward as the purely sequential case. Overall, Decoupled UCT performs best despite its theoretical shortcomings.**

I. Introduction

Monte Carlo Tree Search (MCTS) [1], [2] is a popular search technique that has demonstrated initial success in sequential turn-based games such as Go [3] and Hex [4]. MCTS has also been applied with practical success in general game playing (GGP) [5] and other complex settings [6].

Simultaneous move games are turn-based games where on each turn all the players choose their moves simultaneously. Then, the state of the game in the following turn depends on the collection of moves chosen by all players. A simple example is the one-shot game of Rock, Paper, Scissors where each player has the same three legal moves and wins and losses are determined by the combination of moves chosen by both players. Another example is the classic board game Diplomacy by Avalon Hill, where each turn consists of players submitting lists of orders for each of their units, and the outcome of each order depends on the orders of units in surrounding regions.

The most popular MCTS algorithm is Upper Confidence Bounds for Trees (UCT) [2] based on the bandit algorithm Upper Confidence Bounds (UCB) [7]. Since UCT was originally designed for strictly sequential turn-taking games, its theoretical guarantees (such as eventual convergence to the optimal strategy) do not apply in simultaneous move games [8]. The standard application of UCT to simultaneous move games, first used in general game-playing [9], is a variant we call *Decoupled UCT* (DUCT), where each player uses UCB to select their own moves independent of how the opponent's (simultaneously chosen) move on the same turn can affect the outcome. This variant has been shown to not converge to an optimal strategy, even in a game with a single state [8].

A popular choice for a simultaneous move game among researchers and enthusiasts has been Tron, a game played on a discrete grid inspired by the 1982 movie. In Tron, [10] proposes a purely sequential version which we call *Sequential UCT*. In [11], Sequential UCT in Tron is improved by suggesting several heuristic improvements and handling some of the important simultaneous move situations. Later work proposed several new search variants, including stochastic selection strategies, and compared their success in Tron [12].

Several new selection strategies for simultaneous move MCTS were also proposed for Goofspiel, a domain where playing with randomized ("mixed") strategies is important [13]. Several of these new techniques compute a mixed strategy, and appear to perform better against DUCT in Goofspiel. Under appropriate settings, some of these variants were shown to converge to an optimal strategy [14]. Nonetheless, in Tron, a variant of DUCT outperformed the stochastic variants that were successful in Goofspiel, even though Tron also has situations that require mixed strategies [15]. In addition, the performance of each method in Tron varied significantly based on the board configuration, as first seen in [11]. These results seem to indicate that the success of each simultaneous move MCTS variant may depend largely on the particular game.

In this paper, we aim to give a general overview of the relative performance of the different simultaneous move MCTS variants. First, we describe a general simultaneous move MCTS framework along with several variants used to date. Then, we perform an elaborate analysis of these variants over nine different games chosen from previous work and GGP competitions. The results show that the expected performance, and hence choice of variant, can vary significantly based on the game and opponent.

The structure for the rest of the paper is as follows. In Section II, we describe the background necessary to express our technique. In Section III we describe simultaneous move MCTS. In Section IV, we present the empirical results and discuss them further. Finally, in Section V we conclude and indicate potential future research.

II. Background

In this section, we describe the foundation and terminology upon which our algorithms are based.

A. Monte Carlo Tree Search

Monte Carlo Tree Search (MCTS) [1], [2] is a simulation-based search algorithm often used in games. The main idea is to iteratively run simulations from the root of the search tree to a terminal game state, incrementally growing a tree rooted at the current state of the actual played game.

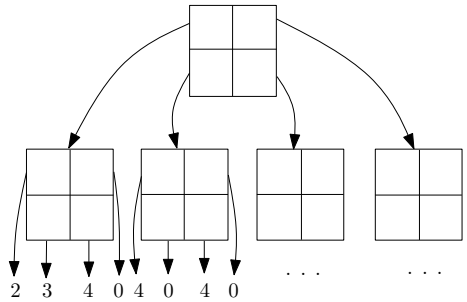

Fig. 1: An example of a two-player simultaneous move game. Each node represents a state of the game. Five nodes are shown, and at each node, each player has two individual moves, for the row player: top (t) and bottom (b), and for the column player: left (l) and right (r), leading to four joint moves per node. Payoffs (u_1) shown are for Player 1, and since $k = 4$, Player 2's payoffs are $u_2 = 4 - u_1$.

MCTS consists of four strategic steps. (1) The *selection step* determines how to traverse the tree from the root node to a leaf node of the tree *L*. It should balance the exploitation of successful moves with the exploration of new moves. (2) In the *playout step*, a random game is simulated from leaf node of the tree *L* until the end of the game. (3) In the *expansion step*, one or more children of *L* are added. (4) In the *back-propagation step*, the reward *R* obtained is back-propagated through the tree from *L* to the root node. Examples and pseudo-code can be found in the MCTS survey article [6].

In simultaneous move games, the process is similar, but simultaneous move selection introduces additional challenges.

B. Simultaneous Move Games

A finite game with simultaneous moves can be described by a tuple $(\mathcal{N}, \mathcal{S} = \mathcal{D} \cup \mathcal{Z}, \mathcal{A}, \mathcal{T}, u_i, s_0)$. The player set $\mathcal{N} = \{1, 2\}$ contains player labels, and by convention a player is denoted by $i \in \mathcal{N}$. $\mathcal{S}$ is a set of states, with $\mathcal{Z}$ denoting the terminal states, and $\mathcal{D}$ the states where players make decisions. $\mathcal{A} = \mathcal{A}_1 \times \mathcal{A}_2$ is the set of joint moves of individual players. We denote $\mathcal{A}_i(s)$ the moves 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 moves for both players. 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. The game begins in an initial state s_0. In our algorithms we assume constant-sum games, so $\forall z \in \mathcal{Z}, u_1(z) + u_2(z) = k$. An example of a constant-sum simultaneous move game with $k = 4$ is depicted in Figure 1.

A *matrix game* is a single step simultaneous move game with legal move 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 (Max) and column c by player 2 (Min). For example, in Matching Pennies, each player has two moves. The row player receives a payoff of 1 if both players choose the same move 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 move set $\mathcal{A}_1(s) \times \mathcal{A}_2(s)$ that either leads to a terminal state or to a subgame which is itself another stacked matrix game.

Define Σ_i to be the set of *behavioral strategies* (where each strategy is a mapping from each state to a probability distribution over actions at that state) for player i. A Nash equilibrium profile in this case is a pair of behavioral strategies optimizing $V^* = \max_{\sigma_1 \in \Sigma_1} \min_{\sigma_2 \in \Sigma_2} \mathbb{E}_{z \sim \sigma}[u_1(z)]$. In other words, neither player can improve their utility by deviating unilaterally. For example, the matrix game that represents the second child (from the left) in Figure 1 has only one equilibrium strategy which mixes equally between both moves, *i.e.,* play with a *mixed strategy* (distribution) of $(0.5, 0.5)$ giving expected payoff $V^* = 2$.

In two-player constant-sum games, a Nash equilibrium strategy is optimal in the minimax sense. It guarantees the payoff of at least V^* against any opponent. Any non-equilibrium strategy has a best response, which will make it win less than V^* in expectation. Moreover, a 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.

Two-player constant-sum simultaneous move games can be solved (*i.e.,* Nash equilibrium strategies computed) using backward induction [16], [17], [18]. The main idea is to start from the endgame positions and individually compute the V^* of each subgame, by solving a linear program at each state, and working back up to the root state. For the example in Figure 1, the value V^* of the matrix game on the bottom-left is 3 (if the row player is maximizing) because each player has a (iteratively) strictly dominated strategy, and V^* of the other game is 2. These are the unique game-theoretic values for a joint moves $\{(l, t), (l, b)\}$ in the parent node. In MCTS, these values are approximated using Monte Carlo sampling, and the variants use the estimates when selecting moves. As in purely sequential games, search techniques approximate these underlying game-theoretic algorithms, but the connection is not as straightforward since the optimal strategies can be mixed and V^* can be any value in the range of $[v_{\min}, v_{\max}]$.

For an elaborate overview of simultaneous move games, including examples and recent work, see [19, Chapter 5].

III. Simultaneous Move MCTS

In this section, we present a generalized description of MCTS based on the one used in Tron [12]. Each specific step (Select, Update, and final move selection) depends on the chosen variant described in the appropriate subsection.

In Simultaneous Move MCTS (SM-MCTS), the main difference with standard MCTS is that during the selection step, at each node, a *joint move* is selected. The convergence to an optimal strategy depends critically on 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,

```
1  SM-MCTS(node s)
2      if s is a terminal state (s ∈ Z) then
3          return u_1(s)
4      else if s ∈ T and EXPANSIONREQUIRED(s) then
5          Choose a previously unselected (a_1, a_2)
6          s' ← T(s, a_1, a_2)
7          Add s' to T
8          u_1 ← PLAYOUT(s')
9          X_s' ← X_s' + u_1
10         n_s' ← n_s' + 1
11         UPDATE(s, a_1, a_2, u_1)
12         return u_1
13     (a_1, a_2) ← SELECT(s)
14     s' ← T(s, a_1, a_2)
15     u_1 ← SM-MCTS(s')
16     UPDATE(s, a_1, a_2, u_1)
17     return u_1
```

Algorithm 1: Simultaneous Move Monte Carlo Tree Search

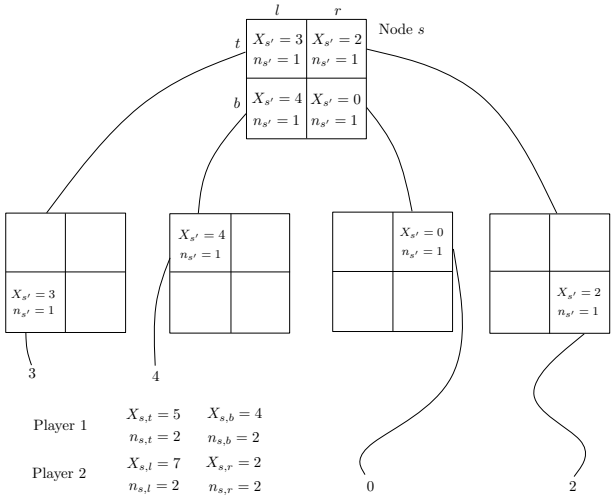

Fig. 2: A detailed example of Decoupled UCT. All payoffs are shown in view of Player 1 (Max).

and a visit count n_s, both initially set to 0. As with standard MCTS, when a state is visited these values are incremented on lines 9 and 10, and in the node updates on lines 11 and 16. Note that Algorithm 1 explicitly stores the rewards and estimates only in view of player one, hence the u_1 notation, and the opponent's reward is obtained as $u_2 = k - u_1$ as necessary. As seen in Figure 1, a matrix of references to the children is maintained at each state.

At node s, the estimated values $\bar{X}_{s'}$ of the children nodes $s' = \mathcal{T}(s, a_1, a_2)$ form an estimated payoff matrix for node s. Each variant below will describe a different way to select a joint move (line 13) and update a node (lines 11 and 16).

We focus on a number of SM-MCTS variants. The most popular one is straightforward adaption of UCT. Though it has been shown to not converge to a Nash equilibrium even in a one-shot single-state game [8], it has nonetheless been popular and successful in general game playing. The second is Exp3 [20], which was also applied successfully to the popular online card game Urban Rivals [21], [22]. Regret Matching, was recently shown to perform well in Goofspiel [13]. The latter two were also used in a recent theoretical analysis of the convergence of SM-MCTS [14]. Finally, we also try Sequential UCT, which has been previously used in Tron [10], [15].

A. Decoupled UCT

In Decoupled UCT (DUCT), each player i maintains for every state s separate reward sums $X^i_{s,a}$ and visit counts $n^i_{s,a}$ for their own move set $a \in \mathcal{A}_i(s)$. In essence, each player *decouples* the matrix in each state and applies UCB over the cumulative rewards for each of their own moves as if there was no joint dependency on these rewards. When a joint move needs to be selected on line 13, each player selects a move that maximizes the decoupled UCB value over their reward estimates independently:

$$a_i = \underset{a \in \mathcal{A}_i(s)}{\operatorname{argmax}} \left\{ \bar{X}^i_{s,a} + C\sqrt{\frac{\ln n_s}{n_{s,a}}} \right\}, \text{ where } \bar{X}^i_{s,a} = \frac{X^i_{s,a}}{n_{s,a}} \tag{1}$$

Here, C is a parameter that determines how much weight to place on exploration. In DUCT, UPDATE simply increments the decoupled values for each player i: $X^i_{s,a_i} \leftarrow X^i_{s,a_i} + u_i$, and $n_{s,a_i} \leftarrow n_{s,a_i} + 1$.

Consider again the game from Figure 1, with player set $\mathcal{N} = \{\text{Max}, \text{Min}\}$, root node s, and move sets $\mathcal{A}_{\text{Max}}(s) = \{t, b\}, \mathcal{A}_{\text{Min}}(s) = \{l, r\}$, $v_{\min} = 0$, and $v_{\max} = 4$. An example of DUCT run for 5 simulations is given in Figure 2. Suppose four simulations are run, $n_s = 4$ and rewards received are $u_{\text{Max}} = 3$ for (t, l), 4 for (b, l), 0 for (b, r), and 2 for (t, r). The tree contains exactly five states: s, and each child of s. On the fifth simulation, Max chooses among t and b and Min chooses among r and l, both by using Equation 1. In this case, $\bar{X}^{\text{Max}}_{s,t} = (3 + 2)/2 = 2.5$ and $\bar{X}^{\text{Max}}_{s,b} = (4 + 0)/2 = 2$. Then $\bar{X}^{\text{Min}}_{s,l} = 4 - ((3 + 4)/2) = 0.5$ and $\bar{X}^{\text{Min}}_{s,r} = 4 - ((2 + 0)/2) = 3$.

In DUCT, after all the simulations have been performed, each player i chooses separately their own final move with the highest estimated reward $\max_{a \in \mathcal{A}_i(s)} \bar{X}^i_{s,a}$.

B. Exp3

In Exp3 [20], 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 own moves. In the case of Exp3, however, $\hat{x}^i_{s,a}$ represents a cumulative *weighted* reward, where each value gets scaled by the probability of it having been sampled. These values and strategies are maintained separately for each player, so Exp3 is decoupled in the same sense as DUCT.

The joint move selected on line 13 is composed of moves independently selected for each player based on a sampling probability distribution composed for each player based on $\hat{x}^i_{s,a}$. The probability of sampling move a_i is

$$\sigma^t_i(s, a_i) = \frac{(1 - \gamma)e^{\eta w^i_{s,a_i}}}{\sum_{a_j \in \mathcal{A}_i(s)} e^{\eta w^i_{s,a_j}}} + \frac{\gamma}{|\mathcal{A}_i(s)|}, \text{ where} \tag{2}$$

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

Here, γ is the probability of exploring (choosing a random move). The update after selecting joint move (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 move was played by the player using $n_{s,a_i} \leftarrow n_{s,a_i} + 1, \hat{x}^i_{s,a_i} \leftarrow \hat{x}^i_{s,a_i} + \frac{u_i}{\sigma^t_i(s,a_i)}$. Dividing the value by the probability of selecting the corresponding move makes $\hat{x}^i_{s,a}$ estimate the sum of rewards over all iterations, not only the ones where a_i was selected.

The mixed strategy used by player i after the simulations are done is given by the frequencies of visit counts of the moves, $\sigma^{final}_i(s, a_i) = n_{s,a_i} \;/\; \sum_{b_i \in A_i(s)} n_{s,b_i}$. Previous work [21] 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

$$n'_{s,a_i} \leftarrow \max\left(0, n_{s,a_i} - \frac{\gamma}{|A_i(s)|} \sum_{b_i \in A_i(s)} n_{s,b_i}\right). \quad (3)$$

For final move selection, a move is chosen by sampling according to a distribution that normalizes n'_{s,a_i}.

C. Regret Matching

This variant applies regret matching [23] 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 node s there is a mixed strategy $\sigma^t_i(s)$ used by each player i for each node s in the tree, initially set to uniform random: $\sigma^0_i(s, a) = 1/|\mathcal{A}(s)|$. Each player i maintains a cumulative regret $r^i_s[a]$ for having played $\sigma^t_i(s)$ instead of $a \in \mathcal{A}_i(s)$. In addition, a table for the average strategy is maintained per player as well $\bar{\sigma}^i_s[a]$. The values in both tables are initially set to 0. As in DUCT, the regret values $r^i_s[a_i]$ are maintained separately by each player. However, the updates and specifically the reward uses a value that is a function of the joint move space.

On iteration t, the selection policy (line 13 in Algorithm 1) first builds the player's current strategies from the cumulative regret. Define the operator $(\cdot)^+ = \max(\cdot, 0)$, *i.e.,* $(-3)^+ = 0$ and $4^+ = 4$, and define

$$\sigma^t_i(s, a) = \frac{r^i_s[a]}{R^+_{sum}} \text{ if } R^+_{sum} > 0 \text{ or } \frac{1}{|\mathcal{A}_i(s)|} \text{ otherwise}, \quad (4)$$

where $R^+_{sum} = \sum_{a \in \mathcal{A}_i(s)} r^{i,+}_s[a]$. The main idea is to adjust the strategy by assigning higher weight proportionally to moves based on the regret of having not taken them over the long-term. To ensure exploration, a γ-on-policy sampling procedure similar to Equation 2 is used choosing move a with probability $\gamma/|\mathcal{A}(s)| + (1 - \gamma)\sigma^t_i(s, a)$.

The updates on line 11 and 16 add regret accumulated at the iteration to the regret tables r^i_s and the average strategy $\bar{\sigma}^i_s[a]$. Suppose joint move (a_1, a_2) is sampled from the selection policy and utility u_i is returned from the recursive call on line 15. 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:

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

and average strategy updates are $\bar{\sigma}^i_s[a] \leftarrow \bar{\sigma}^i_s[a] + \sigma^t_i(s, a)$ for each player. Here, u_1 is the reward for player one and recall that the reward for the other player $u_2 = k - u_1$.

The final move for the root s is chosen by sampling over the strategy obtained by normalizing the values in $\bar{\sigma}^i_s$.

This application of regret matching is similar to Monte Carlo Counterfactual Regret Minimization (MCCFR) [24], except the regret is not counterfactual and the means of child nodes are used to estimate expected values of the subtrees. Convergence to an equilibrium can be guaranteed by backpropagating each node's estimated value but the standard method in Algorithm 1 has been shown to work better in practice [14].

D. Sequential UCT

Sequential UCT (SUCT) performs regular UCT on a *serialized* game tree, *i.e.,* one that is turned into a purely sequential game. In the running example in the simultaneous game, the first simulation is $(t, l) \rightarrow (b, l) \rightarrow 3$. In Sequential UCT, this would be four steps: $t \rightarrow l \rightarrow b \rightarrow l \rightarrow 3$.

In SUCT, a different tree is built, one where a player chooses a move and the opponent is allowed to know which move the player chose and can respond accordingly. The motivation for this is that if a player announces his move to the opponent and the opponent is allowed to choose a 1-ply best response, then the searching player will learn to play the move that has the least chance of being penalized by an opponent. A portion of the serialized tree for the running example game is given in Figure 3.

This serialized tree approach can provide bounds the true minimax value of the root node in the simultaneous move game, and has been applied to RTS combat scenarios [25] and equilibrium computation algorithms [26]. In this paper, we assume that the search player moves and then the opponent responds. This leads to defensive play as explained above. However, this order could be reversed for aggressive play, or even randomized as was done in $\alpha\beta$ search for abstract combat games [27].

IV. Empirical Evaluation

In this section, we discuss our experimental setup for evaluating SM-MCTS. We start by describing nine simultaneous move games used previously in GGP.

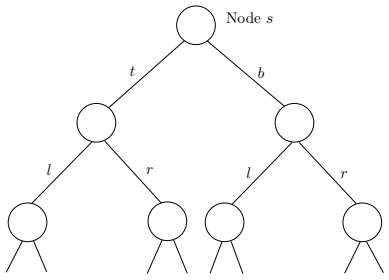

Fig. 3: A Sequential UCT tree after six simulations, with Player 1 as the search player. The root s with the four leaf nodes correspond to the five nodes shown in Figures 1 and 2. In SUCT, Player 2 has two intermediate response nodes.

A. Game Descriptions

Battle is played on an 8×8 board. Each player has 20 disks. These disks can move one square or capture an opponent square next to them. Instead of a move, the player can choose to defend a square occupied by their piece. If an attacker attacks such a defended square, the attacker will be captured. The goal is to be the first player to capture 10 opponent disks.

Bidding Tic-Tac-Toe is a variation of normal Tic-Tac-Toe with a bidding round between normal play that decides who gets to place a marker on the board. Each player begins with three coins, and the X player has an additional tiebreaker token. When a player wins a bidding round, allowing that player to place a marker, the coins used to bid are given to the opponent. The tiebreaker token can optionally be used to break ties, and if so, the tiebreaker token is also given to the opponent. The winning conditions are the same as standard Tic-Tac-Toe.

Chinook is a variant of *Breakthrough* where two independent games are played simultaneously. One game on the white squares and another one on the black squares. Black and White move their pieces simultaneously like Checkers pawns. As in Breakthrough, the first player that reaches the opposite side of the board wins the game.

Goofspiel is a card game where each player gets 13 cards marked 1-13, and there is a face down point-card stack (also 1-13). Every turn, the *upcard* (top card of the point-card stack is turned face up, Each player chooses a *bid* card from their hand simultaneously. The player with the higher bid takes the upcard. The bid cards are then discarded and a new round starts. At the end of 13 rounds, the player with the highest number of points wins, a tie ends in a draw. In this paper, we assume that the point cards have a fixed (decreasing) order.

In *Runners* each turn both players decide how many steps they want to move forward or backward, with a maximum of three steps per turn. The aim is to move 50 steps forward.

Oshi-Zumo(N, K, M) is a wrestling simulation game played on a discrete single-dimensional grid with $2K + 1$ positions, where each player starts with N coins [17]. A wrestler token begins in the middle position. Every turn, each player bids $b \geq M$ coins. The coins bid are then discarded and the player bidding the most coins pushes the wrestler one position closer to the goal for that player. The parameters used in our experiments are $(50, 3, 0)$.

Pawn Whopping is a simultaneous move version of Breakthrough, played on an 8×8 board. Each player has 16 pawns starting on one side of the board and the goal is to move one of their pieces to the other side of the board. Pawns can capture diagonally and only move forward one cell straight. If opposing pawns try to capture each other or move to the same square, no change is made.

Racetrack Corridor is played on a board with length 5 cells and width 3 cells. Each player starts at the top of the board and the aim is to reach the other side of the board before the opponent does. Each move, the player can decide to move forward, sideways or place a wall on the board of the opponent, which spans two thirds of the width of the board.

Tron is a two-player game played on discrete grid possibly obstructed by walls. At each step in Tron both players move to adjacent cells, and a wall is placed in the cells the players started on that turn. Both players try to survive as long as possible. If both players can only move into a wall, can only move off the board or move into each other at the same turn, the game ends in a draw. The initial position is a 13×13 square field with a walled "box" in the center (board (a) from [15]).

	DUCT	SUCT	Exp3	RM
Battle	$C = 0.4$	$C = 0.4$	$\gamma = 0.8$	$\gamma = 0.025$
Bidding Tic-Tac-Toe	$C = 0.0$	$C = 0.0$	$\gamma = 0.9$	$\gamma = 0.150$
Chinook	$C = 0.4$	$C = 0.4$	$\gamma = 0.2$	$\gamma = 0.300$
Goofspiel	$C = 0.4$	$C = 0.8$	$\gamma = 0.1$	$\gamma = 0.200$
Oshi-Zumo	$C = 0.4$	$C = 1.8$	$\gamma = 0.6$	$\gamma = 0.750$
Pawn Whopping	$C = 1.4$	$C = 1.4$	$\gamma = 0.4$	$\gamma = 0.500$
Racetrack Corridor	$C = 1.2$	$C = 0.8$	$\gamma = 0.4$	$\gamma = 0.150$
Runners	$C = 0.8$	$C = 1.8$	$\gamma = 0.3$	$\gamma = 0.500$
Tron	$C = 1.8$	$C = 2.0$	$\gamma = 0.3$	$\gamma = 0.300$

TABLE I: Parameter values.

B. Test Environment

Each of the games described above is implemented in Java. As we intend for our comparison to be general, the domains were influenced by those used in GGP competitions and research work, and our goals are to closely match the conditions of the GGP settings. The advantage is that we do not have to deal with a slow GDL reasoner. Therefore, we can use shorter time settings than the longer ones often used in GGP, and still guarantee more simulations. The rules of the games are implemented without any domain knowledge. Therefore every variant uses a uniform random playout policy. In this way, we still mimic GGP, but the experiments are much faster and generate more games to ensure the assessments are based on data that is statistically significant.

DUCT, SUCT, Exp3, and RM are pair wise compared in each of the nine games. The time setting is 2.5 seconds per move (the number of simulations per second depends on the game and varies between 2200 and 28000). Each combination of one variant versus another in a particular game consists of 1000 games.

In GGP, all utilities are non-negative, so without loss of generality, $u_1, u_2 \geq 0$. Also, games may not be constant-sum. In our implementation, we ensure that all payoffs are appropriately scaled to ensure that they are k-sum, with $k > 0$, by using a similar transformation previously used in CADIAPLAYER [28]: $\text{SCALE}(u_1, u_2) = (\frac{u_1 - u_2 + k}{2}, \frac{u_2 - u_1 + k}{2})$.

C. Parameter Tuning

Before the different algorithms are compared, their parameters are tuned by self-play where different parameter values are compared against a reference parameter. In DUCT, and SUCT the default parameter is $C = 1.4$. The reference parameters for Exp3 and RM are set to $\gamma = 0.2$ and $\gamma = 0.025$, respectively, because they are the optimal parameters in Goofspiel [13]. The tuned parameter values are shown in Table I.

The table shows that the optimal parameter value depends on the game. As it is difficult to choose one parameter value for each algorithm that works well in all games, we choose to use a different parameter value for each game separately. In this way, we can see how well each of the algorithms can perform under optimal conditions. In the GGP context this is difficult, because the game playing agents often have severely restricted time to interpret the rules before play starts.

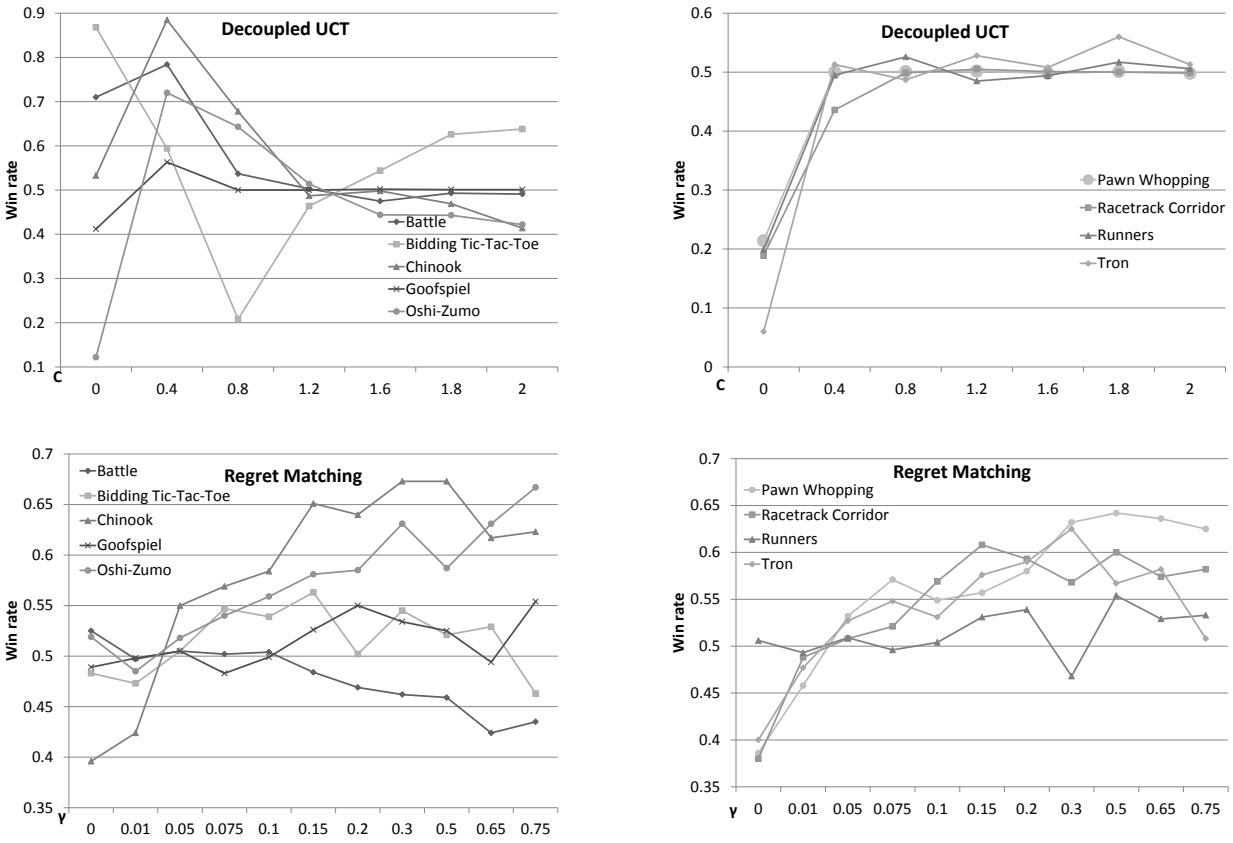

Fig. 4: Tuning Regret Matching and Decoupled UCT

D. Experimental Results

1) Parameter Landscape: Figure 4 shows the win rate of DUCT depending on C against the reference parameter $C = 1.4$ and shows the win rate of Regret Matching depending on γ against the reference parameter $\gamma = 0.025$. The figure highlights two important results. First, it is not possible to choose a parameter value that performs well in all games. Second, the performance of Regret Matching and Decoupled UCT can depend heavily on the parameter value.

We notice that the win rates of DUCT vary quite sensitively in Bidding Tic-Tac-Toe, Oshi-Zumo, and Chinook, and even more so when $C < 1.2$. For Regret Matching, generally interval $\gamma \in [0.2, 0.5]$ seems to be safe but the optimal parameter varies significantly across game types.

The behavior of SUCT is similar to that of DUCT. The behavior of Exp3 is more regular than that of the other variants. Still, optimal parameters varies significantly across game types. Therefore, a separate parameter value is chosen for each game and variant. See Table I for the chosen parameter values.

2) Performance Comparisons: Tables II, III, IV show the performance results for each variant in the games we have tried. Tables report win percentage with 95% confidence intervals. Draws counts as a half win and half loss.

Table II shows the full results of every variant against every other variant in each game. One thing that is clear is that Regret Matching is preferred over all other variants in Oshi-Zumo. However, Regret Matching seems to lose quite significantly to all the other variants in Battle, Chinook, Pawn Whopping, Racetrack Corridor, and Tron, making it a polar choice. Exp3 seems to suffer less in this regard and in some cases (Chinook, Pawn Whopping, Racetrack Corridor, and Tron) beats Regret Matching quite significantly, but generally loses against both DUCT and SUCT in most games.

We also tested another variant of DUCT, called UCB1-Tuned (DUCB1T) [7], which performs almost equally as DUCT. Therefore, the results are not included in the tables. In four games, there seems to be some performance differences between DUCT and DUCB1T. Namely, in Chinook and Tron, DUCB1T performs slightly better than DUCT with a win percentage of 54.30 ($\pm$ 3.05) and 52.90 ($\pm$ 2.35), respectively. There were no significant differences between the performances of DUCT and DUCB1T against Exp3, RM, and SUCT over all games.

We also notice that SUCT is significantly better than DUCT in Tron. We have discovered in previous work that mistakes in Tron are unforgiving [15], so it is possible that the overly defensive play of SUCT more efficiently avoids mistakes that could lead to strong best responses from the opponent. From previous work in Tron, when using informed playout policies UCB1-Tuned selection variant seemed to do

Battle	DUCT	Exp3	RM	SUCT
DUCT		100.00 (± 0.00)	100.00 (± 0.00)	59.90 (± 2.38)
Exp3	0.00 (± 0.00)		42.25 (± 2.79)	0.00 (± 0.00)
RM	0.00 (± 0.00)	57.75 (± 2.79)		0.00 (± 0.00)
SUCT	40.10 (± 2.38)	100.00 (± 0.00)	100.00 (± 0.00)	

B.T.T.T.	DUCT	Exp3	RM	SUCT
DUCT		61.45 (± 2.90)	52.15 (± 3.01)	51.00 (± 2.92)
Exp3	38.55 (± 2.90)		45.65 (± 3.04)	41.15 (± 2.94)
RM	47.85 (± 3.01)	54.35 (± 3.04)		43.85 (± 3.01)
SUCT	49.00 (± 2.92)	58.85 (± 2.94)	56.15 (± 3.01)	

Chinook	DUCT	Exp3	RM	SUCT
DUCT		91.75 (± 1.68)	99.40 (± 0.48)	68.00 (± 2.87)
Exp3	8.25 (± 1.68)		89.45 (± 1.90)	17.95 (± 2.37)
RM	0.60 (± 0.48)	10.55 (± 1.90)		0.65 (± 0.49)
SUCT	32.00 (± 2.87)	82.05 (± 2.37)	99.35 (± 0.49)	

Goofspiel	DUCT	Exp3	RM	SUCT
DUCT		50.15 (± 2.54)	5.00 (± 1.34)	73.85 (± 2.40)
Exp3	49.85 (± 2.54)		37.80 (± 2.98)	58.45 (± 2.82)
RM	95.00 (± 1.34)	62.20 (± 2.98)		30.75 (± 2.83)
SUCT	26.15 (± 2.40)	41.55 (± 2.82)	69.25 (± 2.83)	

O.Z.	DUCT	Exp3	RM	SUCT
DUCT		81.35 (± 1.86)	29.40 (± 2.58)	82.05 (± 1.86)
Exp3	18.65 (± 1.86)		24.20 (± 2.50)	37.00 (± 2.47)
RM	70.60 (± 2.58)	75.80 (± 2.50)		65.10 (± 2.67)
SUCT	17.95 (± 1.86)	63.00 (± 2.47)	34.90 (± 2.67)	

P.W.	DUCT	Exp3	RM	SUCT
DUCT		74.30 (± 1.58)	98.60 (± 0.53)	49.90 (± 0.14)
Exp3	25.70 (± 1.58)		96.30 (± 0.98)	29.00 (± 1.53)
RM	1.40 (± 0.53)	3.70 (± 0.98)		0.85 (± 0.42)
SUCT	50.10 (± 0.14)	71.00 (± 1.53)	99.15 (± 0.42)	

R.C.	DUCT	Exp3	RM	SUCT
DUCT		61.35 (± 1.41)	98.90 (± 0.45)	49.95 (± 0.22)
Exp3	38.65 (± 1.41)		93.90 (± 1.32)	38.20 (± 1.37)
RM	1.10 (± 0.45)	6.10 (± 1.32)		1.40 (± 0.53)
SUCT	50.05 (± 0.22)	61.80 (± 1.37)	98.60 (± 0.53)	

Runners	DUCT	Exp3	RM	SUCT
DUCT		91.10 (± 1.26)	79.80 (± 2.13)	51.45 (± 3.06)
Exp3	8.90 (± 1.26)		38.30 (± 1.39)	10.30 (± 1.31)
RM	20.20 (± 2.13)	61.70 (± 1.39)		20.45 (± 2.14)
SUCT	48.55 (± 3.06)	89.70 (± 1.31)	79.55 (± 2.14)	

Tron	DUCT	Exp3	RM	SUCT
DUCT		59.80 (± 2.54)	77.95 (± 2.31)	46.50 (± 2.40)
Exp3	40.20 (± 2.54)		76.20 (± 2.36)	41.85 (± 2.62)
RM	22.05 (± 2.31)	23.80 (± 2.36)		19.70 (± 2.19)
SUCT	53.50 (± 2.40)	58.15 (± 2.62)	80.30 (± 2.19)	

TABLE II: Performance of each variant vs. each variant in every game. Win rate is for the variant in the listed row.

particularly well [12], [15]. However, in this general context with random playout policies, the advantage of DUCB1T is certainly less clear, as in the original work in Tron [10].

Interestingly, in Goofspiel and Oshi-Zumo, Regret Matching performs significantly better than DUCT. Exp3 also outperforms DUCT in Goofspiel. Again, this is consistent with previous results in Goofspiel [13]. Both Goofspiel and Oshi-Zumo have been solved [17], [18] and their optimal strategies are mixed distributions. In Goofspiel, any deterministic strategy has a best response that exploits it by a lot, so mixing is important. These stochastic selection strategies, which converge to mixed distributions, are likely to be more effective in games where the optimal strategy requires mixing. In the case of Goofspiel, RM wins an outstanding 95.0% of games against DUCT. This shows that the performance of these variants can vary significantly depending on the game type.

A surprising result is that the performance results are not always transitive. For example, in Goofspiel, DUCT beats SUCT, but loses from RM. However, SUCT beats RM in Goofspiel. We believe this effect could be due to algorithms in

Game \ Variant	Exp3	RM	SUCT
Battle	0.00 (± 0.00)	0.00 (± 0.00)	40.10 (± 2.38)
BiddingTicTacToe	38.55 (± 2.90)	47.85 (± 3.01)	49.00 (± 2.92)
Chinook	8.25 (± 1.68)	0.60 (± 0.48)	32.00 (± 2.87)
Goofspiel	49.85 (± 2.54)	**95.00** (± 1.34)	26.15 (± 2.40)
OshiZumo	18.65 (± 1.86)	**70.60** (± 2.58)	17.95 (± 1.86)
PawnWhopping	25.70 (± 1.58)	1.40 (± 0.53)	50.10 (± 0.14)
RacetrackCorridor	38.65 (± 1.41)	1.10 (± 0.45)	50.05 (± 0.22)
Runners	8.90 (± 1.26)	20.20 (± 2.13)	48.55 (± 3.06)
Tron	40.20 (± 2.54)	22.05 (± 2.31)	**53.50** (± 2.40)

TABLE III: Performance playing against DUCT in each game.

	DUCT	SUCT	Exp3	RM
Win Rate	68.34 ± 0.48	63.36 ± 0.50	38.77 ± 0.51	29.54 ± 0.51

TABLE IV: Win rate summary, over all games played.

simultaneous move games being sensitive to the parameters. As these parameters are tuned in self play, it could be that parameters were overfit, because the optimal parameter value for a certain algorithm also depends on the algorithm used by the opponent. Also, due to the importance of mixing in some of the games, it is possible that there could be a "Rock-Paper-Scissors"-like effect of one variant acting as a best response to the other. Therefore, care must be taken when determining which parameter set to use for each search variant. These results suggest that systematic testing against several different variants is prudent in determining robust parameter settings.

Table III shows the summary of the win rates of alternative algorithms against DUCT. This table is important, because DUCT is the standard technique often used in the GGP competitions. Based on these results in Table III it seems that DUCT indeed performs well in most games. This is consistent with the existing successes in general game playing.

A summary of the performance results is given in Table IV. DUCT seems to be a safe choice for these nine games, winning 68.34% of games overall. However, Sequential UCT, winning 63.36% of games, is not too far behind, can be easier to implement, and does seem to perform well against Regret Matching in mixing games such as Goofspiel.

V. Conclusion and Future Work

In this paper, we presented simultaneous move MCTS and several variants, evaluating their performance in nine games used in the general game playing. We investigate several variants: Decoupled UCT, Sequential UCT, Exp3, and Regret Matching. The performance of each variant is evaluated in nine games used in the general game playing literature.

This work leads to three main conclusions. First, the expected behavior of each variant varies with respect to parameter values in simultaneous move MCTS. Finding the correct parameters seems to depend on the game and variant being used, and can vary significantly even against a fixed opponent. This could be explained by the different nature of simultaneous move games and techniques used in search variants. Second, the performance of each variant seems to

depend on both the game and the opponent. In games where mixing is important, then the mixed strategies computed by Regret Matching seem to offer better performance, but in some cases this can even be superseded by Sequential UCT. These intransitive performance rankings suggest that obtaining a robust parameter setting may require more effort than in purely sequential games. Lastly, Decoupled UCT does seem to perform best overall in these games, somewhat justifying its success in general game playing.

For future work, we would like to also measure the effect that the simultaneous move MCTS-Solver [28, Chapter 6] has on these SM-MCTS variants, and test the performance of the different SM-MCTS variants under different circumstances. For example, by changing the time settings, by using a fixed parameter value per variant or by using techniques from general game playing that improve the playouts [5]. It could be interesting to compare performance of SM-MCTS against well-known benchmark players, such as Goofspiel competition strategies [29]. Also, SM-MCTS could be used as a high-level multi-stage planner in real-time strategy games by treating the move of each agent as long-term scripts, extending related previous work in this area [30]. Furthermore, we would like to extend our SM-MCTS variants to games with more than two players, for example as a strategic reasoner module in the seven-player simultaneous move game Diplomacy [31]. Moreover, it would be interesting to investigate how the performance of the methods depends on game complexity, using a synthetic game tree analysis similar to [14]. In addition, we could compare the algorithms on different time settings against optimal players in games that can be solved.

Acknowledgments. This work is partially funded by the Netherlands Organisation for Scientific Research (NWO) in the framework of the projects Go4Nature and GoGeneral, grant numbers 612.000.938 and 612.001.121. Furthermore, we want to thank Hilmar Finnsson for answering our questions about MCTS and GGP.

REFERENCES

[1] R. Coulom, "Efficient selectivity and backup operators in Monte-Carlo tree search," in *5th International Conference on Computers and Games*, ser. LNCS, vol. 4630, 2007, pp. 72–83.

[2] L. Kocsis and C. Szepesvári, "Bandit-based Monte Carlo planning," in *15th European Conference on Machine Learning*, ser. LNCS, vol. 4212, 2006, pp. 282–293.

[3] S. Gelly, L. Kocsis, M. Schoenauer, M. Sebag, D. Silver, C. Szepesvári, and O. Teytaud, "The grand challenge of computer Go: Monte Carlo tree search and extensions," *Communications of the ACM*, vol. 55, no. 3, pp. 106–113, 2012.

[4] B. Arneson, R. B. Hayward, and P. Henderson, "Monte Carlo tree search in Hex," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 2, no. 4, pp. 251–257, 2010.

[5] Y. Björnsson and H. Finnsson, "Cadiaplayer: A simulation-based general game player," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 1, no. 1, pp. 4–15, 2009.

[6] C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton, "A survey of Monte Carlo tree search methods," *IEEE Trans. on Comput. Intel. and AI in Games*, vol. 4, no. 1, pp. 1–43, 2012.

[7] P. Auer, N. Cesa-Bianchi, and P. Fischer, "Finite-time analysis of the multiarmed bandit problem," *Machine Learning*, vol. 47, no. 2/3, pp. 235–256, 2002.

[8] M. Shafiei, N. R. Sturtevant, and J. Schaeffer, "Comparing UCT versus CFR in simultaneous games," in *Proceeding of the IJCAI Workshop on General Game-Playing (GIGA)*, 2009, pp. 75–82.

[9] H. Finnsson, "Cadia-player: A general game playing agent," Master's thesis, Reykjavík University, 2007.

[10] S. Samothrakis, D. Robles, and S. M. Lucas, "A UCT agent for Tron: Initial investigations," in *Proceedings of the 2010 IEEE Conference on Computational Intelligence and Games (CIG)*, 2010, pp. 365–371.

[11] N. G. P. Den Teuling and M. H. M. Winands, "Monte-Carlo Tree Search for the simultaneous move game Tron," in *Proceedings of Computer Games Workshop (ECAI)*, 2012, pp. 126–141.

[12] P. Perick, D. L. St-Pierre, F. Maes, and D. Ernst, "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)*, 2012, pp. 242–249.

[13] M. Lanctot, V. Lisý, and M. H. M. Winands, "Monte Carlo tree search in simultaneous move games with applications to Goofspiel," in *Computer Games*, ser. Communications in Computer and Information Science. Springer, 2014, vol. 408, pp. 28–43.

[14] V. Lisy, V. Kovarik, M. Lanctot, and B. Bosansky, "Convergence of Monte Carlo tree search in simultaneous move games," in *Advances in Neural Information Processing Systems 26*, 2013, pp. 2112–2120.

[15] M. Lanctot, C. Wittlinger, N. G. P. Den Teuling, and M. H. M. Winands, "Monte Carlo tree search for simultaneous move games: A case study in the game of Tron," in *BNAIC 2013*, 2013, pp. 104–111.

[16] S. M. Ross, "Goofspiel — the game of pure strategy," *Journal of Applied Probability*, vol. 8, no. 3, pp. 621–625, 1971.

[17] M. Buro, "Solving the Oshi-Zumo game," in *Advances in Computer Games Conference 10*, ser. IFIP Advances in Information and Communication Technology, vol. 135, 2003, pp. 361–366.

[18] G. C. Rhoads and L. Bartholdi, "Computer solution to the game of pure strategy," *Games*, vol. 3, no. 4, pp. 150–156, 2012.

[19] A. Saffidine, "Solving games and all that," Ph.D. dissertation, Université Paris-Dauphine, Paris, France, 2013.

[20] P. Auer, N. Cesa-Bianchi, Y. Freund, and R. E. Schapire, "Gambling in a rigged casino: The adversarial multi-armed bandit problem," in *Proceedings of the 36th Annual Symposium on Foundations of Computer Science*, 1995, pp. 322–331.

[21] O. Teytaud and S. Flory, "Upper confidence trees with short term partial information," in *Applications of Eolutionary Computation (EvoApplications 2011), Part I*, ser. LNCS, vol. 6624, 2011, pp. 153–162.

[22] D. L. St-Pierre, Q. Louveaux, and O. Teytaud, "Online sparse bandit for card games," in *Advances in Computer Games*, ser. LNCS, vol. 7168, 2012, pp. 295–305.

[23] S. Hart and A. Mas-Colell, "A simple adaptive procedure leading to correlated equilibrium," *Econometrica*, vol. 68, no. 5, pp. 1127–1150, 2000.

[24] M. Lanctot, K. Waugh, M. Zinkevich, and M. Bowling, "Monte carlo sampling for regret minimization in extensive games," in *Advances in Neural Information Processing Systems 22*, 2009, pp. 1078–1086.

[25] D. Churchill, A. Saffidine, and M. Buro, "Fast heuristic search for RTS game combat scenarios," in *8th AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE)*, 2012, pp. 112–117.

[26] B. Bosansky, V. Lisy, J. Cermak, R. Vitek, and M. Pechoucek, "Using double-oracle method and serialized alpha-beta search for pruning in simultaneous moves games," in *IJCAI 2013*, 2013, pp. 48–54.

[27] A. Kovarsky and M. Buro, "Heuristic search applied to abstract combat games," in *Proceedings of the The Eighteenth Canadian Conference on Artificial Intelligence*, ser. LNCS, vol. 3501, 2005, pp. 66–78.

[28] H. Finnsson, "Simulation-based general game playing," Ph.D. dissertation, Reykjavík University, 2012.

[29] M. Dror and G. Kendall, "Repeated Goofspiel: A game of pure strategy," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 4, no. 2, pp. 312–324, 2013.

[30] F. Sailer, M. Buro, and M. Lanctot, "Adversarial planning through strategy simulation," in *IEEE Symposium on Computational Intelligence and Games (CIG)*, 2007, pp. 37–45.

[31] A. Fabregues and C. Sierra, "DipGame: A challenging negotiation testbed," *Engineering Applications of Artificial Intelligence*, vol. 24, no. 7, pp. 1137–1146, 2011.