Sequential Halving for Partially Observable Games

Tom Pepels[1(✉)], Tristan Cazenave[2], and Mark H.M. Winands[1]

[1] Department of Data Science and Knowledge Engineering, Maastricht University, Maastricht, The Netherlands
{tom.pepels,m.winands}@maastrichtuniversity.nl
[2] LAMSADE - Université Paris-Dauphine, Paris, France
cazenave@lamsade.dauphine.fr

Abstract. This paper investigates Sequential Halving as a selection policy in the following four partially observable games: Go Fish, Lost Cities, Phantom Domineering, and Phantom Go. Additionally, H-MCTS is studied, which uses Sequential Halving at the root of the search tree, and UCB elsewhere. Experimental results reveal that H-MCTS performs the best in Go Fish, whereas its performance is on par in Lost Cities and Phantom Domineering. Sequential Halving as a flat Monte-Carlo Search appears to be the stronger technique in Phantom Go.

1 Introduction

Partially observable games introduce the complexity of uncertainty in gameplay. In partially observable games, some element of the game is not directly observable. The unknown element can be introduced by hiding certain parts of the current state to the player (e.g., hiding the rank of piece in Stratego), in game theory this is also called imperfect information. Other than in fully observable games, we cannot directly search for sequences of actions leading to promising moves using the partially visible state. In this paper we discuss four different partially observable games: Go Fish and Lost Cities, which are card games with imperfect information, and the so-called phantom games: Phantom Domineering and Phantom Go.

Different approaches have been suggested for handling partial observability in Monte-Carlo Tree Search (MCTS) in such domains. Such as Determinized UCT [13] where a random game state is sampled before the search (*i.e.,* determinized), and multiple trees are maintained per determinization. The recently introduced Information Set MCTS [13] maintains information sets of states reachable in the current determinization in the tree, as such re-using statistics over multiple determinizations in the tree.

In this paper we investigate the effects of using Sequential Halving [16] as a selection policy in partially observable games. We continue to study the Hybrid MCTS [20] algorithm, introduced as a method of minimizing simple and cumulative regret simultaneously during search.

T. Cazenave et al. (Eds.): CGW 2015/GIGA 2015, CCIS 614, pp. 16–29, 2016.
DOI: 10.1007/978-3-319-39402-2_2

The paper is structured as follows. First, in Sect. 2, we give a brief overview of MCTS. Next, in Sect. 3 we discuss Sequential Halving, and how it may be applied to MCTS in partially observable games. After this we describe the test domains in Sect. 4. Finally, we show the experimental results in Sect. 5, and discuss our conclusions and directions for future research in Sects. 6 and 7.

2 Monte-Carlo Tree Search

Monte-Carlo Tree Search (MCTS) is a best-first search method based on random sampling by Monte-Carlo simulations of the state space of a domain [12,17]. In game play, this means that decisions are made based on the results of randomly simulated play-outs. MCTS has been successfully applied to various turn-based games such as Go [22], Lines of Action [26], and Hex [1]. Moreover, MCTS has been used for agents playing real-time games such as the Physical Traveling Salesman [21], real-time strategy games [4], and Ms Pac-Man [19], but also in real-life domains such as optimization, scheduling, and security [6].

In MCTS, a tree is built incrementally over time, which maintains statistics at each node corresponding to the rewards collected at those nodes and number of times they have been visited. The root of this tree corresponds to the current position. MCTS performs iteratively simulations until a computational threshold is reached, *i.e.,* a set number of simulations, an upper limit on memory usage, or a time constraint.

Each MCTS simulation consists of two main steps, (1) the *selection* step, where moves are selected and played inside the tree according to the selection policy until a leaf is *expanded*, and (2) the *play-out*, in which moves are played according to a simulation policy, outside the tree. At the end of each play-out a terminal state is reached and the result is *back-propagated* along the selected path in the tree from the expanded leaf to the root.

2.1 UCT

During the selection step, a policy is required to explore the tree to decide on promising options. For this reason, the Upper Confidence Bound applied to Trees (UCT) [17] was derived from the UCB1 [3] policy. In UCT, each node is treated as a multi-armed bandit problem whose arms are the moves that lead to different child nodes. UCT balances the exploitation of rewarding nodes whilst allowing exploration of lesser visited nodes. Consider a node p with children $I(p)$, then the policy determining which child i to select is defined as:

$$i^* = argmax_{i \in I(p)} \left\{ v_i + C\sqrt{\frac{\ln n_p}{n_i}} \right\}, \tag{1}$$

where v_i is the score of the child i based on the average result of simulations that visited it, n_p and n_i are the visit counts of the current node and its child, respectively. C is the exploration constant to tune. UCT is applied when the visit count of p is above a threshold T, otherwise a child is selected at random. UCB1 and consequently, UCT incorporate both exploitation and exploration.

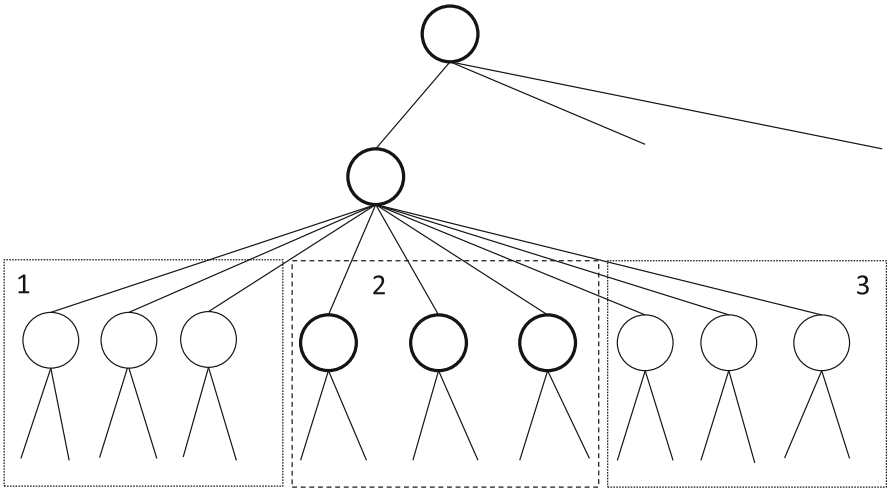

Fig. 1. Example of three determinizations within a single tree. The selected determinization is 2. All unreachable nodes in determinization 2 will not be selected.

2.2 MCTS in Partially Observable Games

To deal with games having imperfect information, determinization can be applied in the MCTS engine. The principle behind determinization is that, at the start of each simulation at the root, the hidden information is 'filled in', ensuring it is consistent with the history of the current match.

Determinization has been called "averaging over clairvoyance" [23], where players never try to hide or gain information, because in each determinization, all information is already available. Despite these shortcomings, it has produced strong results in the past, for instance in Monte-Carlo engines for the trick-based card game Bridge [15], the card game Skat [8], Scrabble [24], and Phantom Go [9].

Determinization in the MCTS framework has been applied in games such as Scotland Yard [18] and Lord of the Rings: The Confrontation [13]. It works as follows. For each MCTS simulation starting at the root the missing information is filled in a random manner. The determinization is used throughout the whole simulation. Next, there are two approaches to build and traverse the search tree.

The first approach is by generating a separate tree for each determinization [13]. After selecting a determinization at the root node, the corresponding tree is traversed. Based on majority voting [18] the final move can be selected. Each candidate move receives one vote from each tree where it is the move that was played most often. The candidate move with the highest number of votes is selected as the best move. If more moves are tied, the move with the highest number of visits over all trees is selected. The concept of separate-tree determinization is similar to root parallelization.

The second approach is using single-tree determinization [11,13,18]. When generating the tree, all possible moves from all possible determinizations are generated. When traversing the tree, only the moves consistent with the current

determinization are considered. An example is given in Fig. 1. The advantage of this technique is that information is shared between different determinizations, increasing the amount of usable information. This type of determinization is also named Single-Observer Information Set Monte-Carlo Tree Search [13].

3 Sequential Halving and MCTS in Partially Observable Games

In this section we describe our approach to applying Hybrid MCTS [20] (H-MCTS) to partially observable games. H-MCTS is based on the concept of minimizing *simple regret* near the root, and *cumulative regret* in the rest of the tree. Simple regret is defined as the regret of not *recommending* the optimal move. Whereas cumulative regret is the sum over the regret of having *selected* suboptimal moves during sampling.

In their analysis of the links between simple and cumulative regret in multi-armed bandits, Bubeck *et al.* [7] found that upper bounds on cumulative regret lead to lower bounds on simple regret, and that the smaller the upper bound on the cumulative regret, the higher the lower bound on simple regret, regardless of the recommendation policy, *i.e.,* the smaller the cumulative regret, the larger the simple regret. As such, no policy can give an optimal guarantee on both simple and cumulative regret at the same time. Since UCB gives an optimal upper bound on cumulative regret, it cannot also provide optimal lower bounds on simple regret. Therefore, a combination of different regret minimizing selection methods in the same tree is used in H-MCTS.

This section is structured as follows, first we discuss Sequential Halving, a novel simple regret minimizing algorithm for multi-armed bandits, in Subsect. 3.1. Next, in Subsect. 3.2 we discuss how a hybrid search technique may be used in partially observable games.

3.1 Sequential Halving

Non-exploiting selection policies have been proposed to decrease simple regret at high rates in multi-armed bandits. Given that UCB1 [3] has an optimal rate of cumulative regret convergence, and the conflicting limits on the bounds on the regret types shown in [7], policies that have a higher rate of exploration than UCB1 are expected to have better bounds on simple regret. Sequential Halving (SH) [16] is a novel, pure exploration technique developed for minimizing simple regret in the multi-armed bandit (MAB) problem.

In many problems there are only one or two good decisions to be identified, this means that when using a pure exploration technique, a potentially large portion of the allocated budget is spent sampling suboptimal arms. Therefore, an efficient policy is required to ensure that inferior arms are not selected as often as arms with a high reward. Successive Rejects [2] was the first algorithm to show a high rate of decrease in simple regret. It works by dividing the total computational budget into distinct rounds. After each round, the single worst

Algorithm 1. Sequential Halving [16].

Input: total budget T, K arms
Output: recommendation J_T

$S_0 \leftarrow \{1, \ldots, K\}$, $B \leftarrow \lceil \log_2 K \rceil - 1$
for *k=0* **to** B **do**
    sample each arm $i \in S_k$, $n_k = \left\lfloor \frac{T}{|S_k| \lceil \log_2 |S| \rceil} \right\rfloor$ times
    update the average reward of each arm based on the rewards
    $S_{k+1} \leftarrow$ the $\lceil |S_k|/2 \rceil$ arms from S_k with the best average
return *the single element of* S_B

arm is removed from selection, and the algorithm is continued on the reduced subset of arms. Sequential Halving [16], was later introduced as an alternative to Successive Rejects, offering better performance in large-scale MAB problems.

SH divides search time into distinct rounds, during each of which, arms are sampled uniformly. After each round, the empirically worst half of the remaining arms are removed until a single one remains. The rounds are equally distributed such that each round is allocated approximately the same number of trials (budget), but with smaller subset of available arms to sample. SH is detailed in Algorithm 1.

3.2 Hybrid MCTS for Partially Observable Games

Hybrid MCTS (H-MCTS) has been proposed by Pepels *et al.* in [20]. The technique uses recursive Sequential Halving, or SHOT [10] to minimize simple regret near the root as depicted in Fig. 2. The hybrid technique has shown to improve performance in several domains, including Amazons, Atari Go and Breakthrough. Previous algorithms that use MCTS with simple regret minimizing selection methods showed similar improvements in recommended moves in Markov Decision Processes [14,25].

In this paper we apply H-MCTS to partially observable games. The problem with these domains is that, when using multiple determinizations during search, revisiting nodes may result in different playable moves. This is not a problem when using selection methods such as UCT, which are greedy and select moves based on the current statistics. However, because SH is a uniform exploration method, in order to guarantee its lower bound on simple regret it must be able to revisit the same node a predetermined number of times. In other words, available moves should not change in between visits of the algorithm, or its specifically designed budget allocation is no longer valid.

In all partially observable games, the current player always has knowledge over the current set of moves that he can play given a fully observable determinization. Therefore, at the root of the search tree, moves are consistent between visits. As such, SH can be used to uniformly explore moves at the root without problems. When using multiple determinizations in a single tree, as in IS-MCTS, however, it is no longer possible to use SH deeper in the tree.

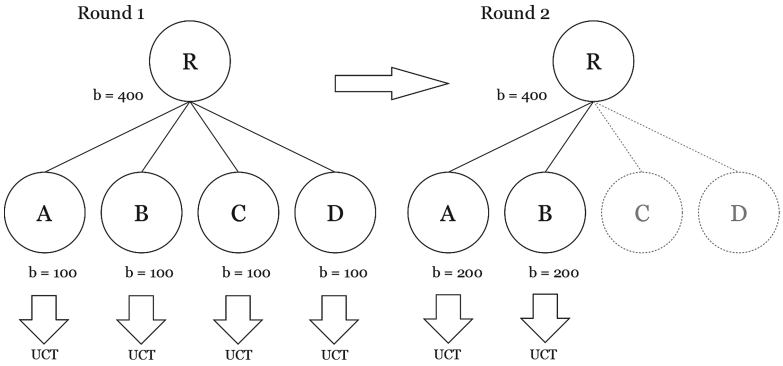

Fig. 2. Example rounds of H-MCTS with a budget limit $B = 150$. Sequential Halving is applied only at the root. On all other plies, UCT in the form of IS-MCTS is applied.

Each time a node is visited it may have a different subset of children based on the determinization (as depicted in Fig. 1). However, when using determinized UCT with a finite set of individual trees per determinization, SH can be used to select nodes deeper than the root, such an investigation is a possible direction for future research.

The approach is detailed in Algorithm 2. At the root, budget is allocated according to SH. For each sample, the appropriate IS-MCTS implementation can be used [13]. For this paper, based on the test domains (Sect. 4), we use single observer IS-MCTS.

4 Test Domains

In this section we discuss the partially observable games which are used in the experiments in Sect. 5. First, we describe the two card games: Go Fish and Lost Cities. Next, the phantom games Phantom Domineering and Phantom Go are explained.

Algorithm 2. Sequential Halving and Information Set MCTS.

Input: total budget T, K moves
Output: recommendation J_T

$S_0 \leftarrow \{1, \ldots, K\}$, $B \leftarrow \lceil \log_2 K \rceil - 1$
for *k=0* **to** B **do**
  for *each move* $i \in S_k$ **do**
    $n_k \leftarrow \left\lfloor \frac{T}{|S_k| \lceil \log_2 |S| \rceil} \right\rfloor$
    for *n=0* **to** n_k **do**
      select a new determinization d at random
      sample move i using IS-MCTS and determinization d
      update the average of i reward based on the sample
  $S_{k+1} \leftarrow$ the $\lceil |S_k|/2 \rceil$ moves from S_k with the best average
return *the single element of* S_B

4.1 Card Games

In both Go Fish and Lost Cities, cards are drawn from a randomly shuffled deck, limiting the possible predictions of future states. Moreover, in both games, moves available to the opponent are either partially or completely invisible. However, whenever a move is made, it becomes immediately known to both players. As these games progress, more information regarding the actual game state becomes available to both players.

Go Fish is a card game which is generally played by multiple players. The goal is to collect as many 'books' of 4 cards of equal rank. All players hide their cards from each other, and only finished books of four cards are placed face-up on the table. Each turn, a player may ask a single other player for a specific rank. If the questioned player has any cards of the requested rank in his hand, he gives them to the requesting player, which may consequently make a new request. If the questioned player does not possess a card of the requested rank, the questioning player must 'go fish', drawing a card from the stack, and the turn moves to the next player. The game ends when there are no more cards on the stack, and the player with the most finished books wins the game.

In our implementation, the game was slightly modified to allow it to be played by two players. Both players receive seven cards in hand at the start of the game. Moreover, the finished books are not similarly rewarded. Books of numbered cards give a score of one, whereas books of face cards assign a score of two, a book of aces gives a score of three. As a result, when the game ends, the player with the highest score wins.

The game state is determinized by removing from the non-visible player's all card drawn from the deck, shuffling the deck and re-drawing the non-visible player's hand. This means that whenever a card was obtained from the opponent it is no longer treated as invisible, because it cannot be anywhere else than in the opponent's hand or visible on the table in a finished book.

Lost Cities is a two-player card game, designed in 1999 by Reiner Knizia. The goal of the game is to achieve the most profitable set of expeditions to one or more of five lost cities. Players start expeditions by placing numbered cards on them, each player can start up to five expeditions regardless of the opponent's expeditions. Each card in the game has a color and a number, the colors represent one of the five expeditions, the numbers representing the score gained. Next to these cards, colored investment cards cumulatively double the score awarded for an expedition. The deck consists of 60 cards, nine numbered cards per color, and three investment cards per color.

Placing a card on an empty expedition 'initializes' it with a cost of 20. Or, when an investment card is played, with a score of $20 \times I_c$, where I_c is the number of investment cards played on expedition c. These cards can only be played on an expedition when no other cards have been played on it. For example, playing the 'red 5' card starts the red expedition with a cost of 20 and a score of 5 resulting in a -15 score for the player. With a single investment card on this expedition, the score will be 30. Playing more cards on the expedition leads to higher scores. However, only increasing cards may be placed on top of others. In this example, the card 'red 3' can no longer be played, whereas the 'red 8' card can be played.

Each turn, players may either play or discard a card, and draw a card from the draw pile or one of the discard piles. Discarding a card places it on top of one of the colored discard piles which are accessible to both players. The game ends when no cards are left on the draw pile, the player with the highest score wins the game.

In Lost Cities, interaction between players is limited. However, players have to carefully choose their expeditions partly based on their opponent's choices. Moreover, players must be careful not to discard cards which may benefit their opponent, but at the same time take care that they can draw cards beneficial to their chosen expeditions.

As in Go Fish, the game state is determinized by removing the non-visible player's hand, shuffling the deck and re-drawing the non-visible player's hand.

4.2 Phantom Games

Next, we describe two so-called phantom games, Phantom Domineering and Phantom Go. Phantom games are modified versions of fully observable games, in which part of the game state is made invisible to the players. Both games are otherwise fully deterministic, *i.e.,* no roll of the dice, or drawing cards. Consequently, whenever a player makes a move it may be rejected, the player may move again until his move is no longer rejected. Playing a move that is rejected is always beneficial, since it provides the player with new information of the actual game state.

Phantom Domineering is based on the combinatorial game Domineering, which is generally played on a square board with two players. Each turn players block two adjacent positions on the board, one player plays vertically, and the other horizontally. The game ends when one of the players cannot make move. As with most combinatorial games, the first player unable to make a move loses the game, and draws are not possible.

In Phantom Domineering, players can only directly observe their own pieces on the board. For both players, their opponent's pieces are hidden, and can only be observed indirectly by performing rejected moves. A unique property in Phantom Domineering is that rejected moves do not provide immediate information about the opponent's moves. In games where moves consist of occupying single positions, a rejected move can immediately reveal an opponent's move. In Phantom Domineering, however, a rejected move means that either one of the two positions is blocked, or both. Therefore, when determinizing, all opponent's stones are first replaced such that they match the rejected moves, after this, all remaining stones are placed randomly on the board.

Phantom Go is a version of Go played in which the opponent's stones are not revealed. When a move is illegal it is usually because there is an opponent's stone on the chosen intersection. In this case a referee publicly announces that the player made an illegal move and the same player may move again. The Chinese rules are used for scoring games. Phantom Go is played by humans at Go congresses and is enjoyed by spectators who can see both players' boards as well as the complete referee board.

During determinization opponent stones are placed on illegal moves. The remaining opponent stones are placed randomly on the determinized board [9]. The principle of our engine, GoLois, is to perform one play-out per determinization. For each possible move, a large number of determinizations followed by play-outs is performed. The move with the highest average is then chosen. Using this approach, GoLois won the gold medal in 5 of the 6 Phantom Go tournaments held during the last Computer Olympiads.

5 Experiments and Results

In this section we show the results of the experiments performed on four, partially observable two-player games. H-MCTS and the games were implemented in two different engines. Go Fish, Lost Cities and Phantom Domineering are implemented in a Java based engine. Phantom Go is implemented in the *C++* based engine GoLois.

Lost Cities relies heavily on a heuristic play-out strategy which prevents obvious bad moves such as starting an expedition without a chance of making a profit. These heuristics improve play over a random play-out by up to 40 %.

Table 1. Win rates with respect to the row player. Minimum of 1,000 games per experiment, 10,000 simulations per move.

	Go Fish			
	H-MCTS	SH	MCTS	UCB
H-MCTS	-	**60.9 %** ± 2.9	**54.3 %** ± 1.9	**62.3 %** ± 2.9
SH	39.1 % ± 2.9	-	44.0 % ± 3.0	51.3 % ± 2.0
MCTS	45.7 % ± 1.9	**56.0 %** ± 3.0	-	**55.0 %** ± 3.1
UCB	37.7 % ± 2.9	48.7 % ± 2.0	45.0 % ± 3.1	-
	Lost Cities			
	H-MCTS	SH	MCTS	UCB
H-MCTS	-	46.1 % ± 3.1	**54.1 %** ± 3.1	47.1 % ± 3.1
SH	**53.9 %** ± 3.1	-	**55.6 %** ± 1.9	50.1 % ± 1.9
MCTS	45.9 % ± 3.1	44.4 % ± 1.9	-	45.3 % ± 3.1
UCB	52.9 % ± 3.1	49.9 % ± 1.9	**54.7 %** ± 3.1	-
	8 × 8 Phantom Domineering			
	H-MCTS	SH	MCTS	UCB
H-MCTS	-	45.1 % ± 3.1	**59.9 %** ± 3.0	**59.5 %** ± 3.0
SH	**54.9 %** ± 3.1	-	**55.1 %** ± 3.1	**58.6 %** ± 3.1
MCTS	41.1 % ± 3.0	44.9 % ± 3.1	-	49.4 % ± 3.1
UCB	40.5 % ± 3.0	41.4 % ± 3.1	51.6 % ± 3.1	-

Table 2. Win rates with respect to the row player. Minimum of 1,000 games per experiment, 25,000 simulations per move.

	Go Fish			
	H-MCTS	SH	MCTS	UCB
H-MCTS	-	**62.2 %** ± 2.9	**55.2 %** ± 3.0	**61.8 %** ± 2.9
SH	42.2 % ± 3.0	-	42.2 % ± 3.0	51.7 % ± 3.1
MCTS	44.9 % ± 3.0	**57.9 %** ± 3.0	-	**59.0 %** ± 3.0
UCB	38.2 % ± 2.9	48.3 % ± 3.1	41.0 % ± 3.1	-
	Lost Cities			
	H-MCTS	SH	MCTS	UCB
H-MCTS	-	48.6 % ± 1.9	**52.7 %** ± 1.9	44.9 % ± 3.0
SH	51.4 % ± 1.9	-	**57.6 %** ± 3.1	52.8 % ± 3.1
MCTS	47.4 % ± 1.9	42.4 % ± 3.0	-	43.7 % ± 1.9
UCB	**55.1 %** ± 3.1	47.3 % ± 3.1	**56.3 %** ± 1.9	-
	8 × 8 Phantom Domineering			
	H-MCTS	SH	MCTS	UCB
H-MCTS	-	48.9 % ± 3.1	53.0 % ± 3.1	**54.5 %** ± 3.1
SH	51.1 % ± 3.1	-	**56.1 %** ± 3.1	51.8 % ± 3.1
MCTS	47.0 % ± 3.1	43.9 % ± 3.1	-	51.3 % ± 3.1
UCB	45.6 % ± 3.1	48.7 % ± 3.1	51.3 % ± 3.1	-

In Phantom Domineering, an ϵ-greedy play-out strategy selects moves based on the number of available moves for the opponent and the player to move. It chooses the move that maximizes their difference. For both Go Fish and Phantom Go, moves are selected uniformly random during play-outs.

In the next subsection, we run experiments on the test domains using a set of different algorithms:

- **H-MCTS** selects moves according to Sequential Halving at the root and UCT in all other parts of the tree, according to Algorithm 2. In all domains, single observer IS-MCTS [13] is used.
- **SH** selects among available moves according to Sequential Halving (Algorithm 1), and samples the moves by play-out immediately. As such, no search is performed.
- **MCTS** selects moves using UCT from root to leaf. As in H-MCTS, single observer IS-MCTS is used.
- **UCB** selects among available moves according to the UCT selection method (Eq. 1) and samples the move immediately by play-out. As such, no search is performed. The method is similar to using the UCB1 algorithm for MABs.

Table 3. Experimental results for Phantom Go. SH vs. UCB with varying C constant. 1,000 games, win rates with respect to SH.

	C	Phantom Go	
		10,000 Simulations	25,000 Simulations
SH vs. UCB	0.1	**69.6 %** ± 2.9	**70.6 %** ± 2.9
	0.2	**58.8 %** ± 3.1	**58.6 %** ± 3.1
	0.3	**58.1 %** ± 3.1	**54.3 %** ± 3.1
	0.4	**58.0 %** ± 3.1	53.0 % ± 3.1
	0.5	**57.1 %** ± 3.1	**55.3 %** ± 3.1
	0.6	**59.1 %** ± 3.1	51.9 % ± 3.1
	0.7	**60.3 %** ± 3.1	**56.7 %** ± 3.1
	0.8	**61.5 %** ± 3.1	**58.6 %** ± 3.1
	0.9	**64.5 %** ± 3.0	**57.8 %** ± 3.1

In all experiments, and for all algorithms, a new determinization is uniformly selected for each simulation. For each individual game, the C constant, used by UCT (Eq. 1) was tuned. MCTS, UCB, and H-MCTS use the same value for the C constant in all experiments.

5.1 Results

For each table, the results are shown with respect to the row algorithm, along with a 95 % confidence interval. For each experiment, the players' seats were swapped such that 50 % of the games are played as the first player, and 50 % as the second, to ensure no first-player or second-player bias. Because H-MCTS cannot be terminated any-time we present only results for a fixed number of simulations. In each experiment, both players are allocated a budget of either 10,000, or 25,000 play-outs. In all tables, significantly positive results are bold-faced.

Tables 1 and 2 show the comparative results for search performed with 10,000 and 25,000 simulations per move, respectively. For most experiments in these tables, 1,000 games were played. However, in some cases where results were close to confidence bounds, 1,500 extra games were played. First, results show that only in Go Fish did performing search improve performance over flat Monte-Carlo sampling, in both Lost Cities and Phantom Domineering performing search did not improve performance. This coincides with previous results for Phantom Go, for which it was determined that search could did not perform better than UCB sampling.

In all games, using H-MCTS improves performance over MCTS when sampling 10,000 simulations per move. In the 25,000 case, MCTS and UCB's performances appear to recover in Lost Cities and Phantom Go. In Go Fish,

performance is stable with respect to the number of simulations. For the games where performing search does not improve performance over single-ply sampling, SH is either on par or outperforms UCB.

In all cases, in both experimental setups, SH or H-MCTS either outperforms MCTS and UCB significantly, or does not negatively impact performance. In Phantom Domineering, sampling using SH improves performance over UCB by up to 8.6 %. A significant improvement when considering that no knowledge or heuristics were introduced in the search. Moreover, SH improves the performance of the award-winning engine GoLois by up to 7.1 % over UCB, as shown in Table 3. In this table we detail the results over different C constants for UCB, showing that without tuning any parameter, SH is able to outperform UCB in all cases. UCB's performance similarly somewhat recovers when given more simulations. However, in all but two cases (when $C = 0.4$ or $C = 0.6$), SH still significantly outperforms UCB with 25,000 simulations per move.

6 Conclusions

This paper has investigated Sequential Halving as a selection policy in partially observable games. In the MCTS framework, Sequential Halving was applied at the root of the search tree, and UCB1 elsewhere, leading to a hybrid algorithm called H-MCTS. Experimental results revealed that H-MCTS performed the best in Go Fish, whereas its performance is on par in Lost Cities and Phantom Domineering. In Phantom Go, Sequential Halving as a flat Monte-Carlo Search was the best algorithm for 10,000 play-outs. For 25,000 play-outs, it was still competitive but the difference with the alternative approach UCB was not statistically significant. Even in cases where Sequential Halving was not better it still has the advantage that it is parameter free.

A possible cause for concern when using UCT in partially observable domains is that the statistics in the tree may become conditioned on a set of determinizations. When a new determinization is used for each sample, the current statistics of each node are biased towards previous determinizations and may not necessarily hold for other determinizations in the future. A uniform selection method such as Sequential Halving may circumvent this possible problem, since selection is not based on the current statistics of each node. Rather, nodes are explored uniformly regardless of their statistics and are only removed from selection after being sampled equally often as their siblings.

7 Future Research

Based on the results in this paper and previous work [20], H-MCTS and Sequential Halving have shown promising result in both fully and partially observable games. This leads to several directions for future research. We propose an investigation into SHOT and H-MCTS in partially observable games by using a limited set of determinizations, and a single tree per determinization. Because in these cases it is possible to use Sequential Halving at internal nodes other than the

root. For future work in H-MCTS in general, the All-Moves-As-First (AMAF) [5] heuristic is considered, a popular method used in MCTS to improve early estimation of nodes. For partially observable domains in specific, we intend to investigate non-uniform selection of determinizations.

References

1. Arneson, B., Hayward, R., Henderson, P.: Monte-Carlo tree search in Hex. IEEE Trans. Comput. Intell. AI Games **2**(4), 251–258 (2010)
2. Audibert, J., Bubeck, S., Munos, R.: Best arm identification in multi-armed bandits. In: Proceedings of the 23rd Conference on Learning Theory, pp. 41–53 (2010)
3. Auer, P., Cesa-Bianchi, N., Fischer, P.: Finite-time analysis of the multiarmed bandit problem. Mach. Learn. **47**(2–3), 235–256 (2002)
4. Balla, R.K., Fern, A.: UCT for tactical assault planning in real-time strategy games. In: Boutilier, C. (ed.) Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI), pp. 40–45 (2009)
5. Bouzy, B., Helmstetter, B.: Monte-Carlo Go developments. In: van den Herik, H.J., Iida, H., Heinz, E.A. (eds.) Advances in Computer Games. IFIP, vol. 135, pp. 159–174. Springer, New York (2004)
6. Browne, C., 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 Trans. Comput. Intell. AI Games **4**(1), 1–43 (2012)
7. Bubeck, S., Munos, R., Stoltz, G.: Pure exploration in finitely-armed and continuous-armed bandits. Theor. Comput. Sci. **412**(19), 1832–1852 (2010)
8. Buro, M., Long, J., Furtak, T., Sturtevant, N.: Improving state evaluation, inference, and search in trick-based card games. In: Boutilier, C. (ed.) Proceedings of the 21st International Joint Conference on Artificial Intelligence, IJCAI 2009, Pasadena, CA, USA, pp. 1407–1413 (2009)
9. Cazenave, T.: A phantom-go program. In: van den Herik, H.J., Hsu, S.-C., Hsu, T., Donkers, H.H.L.M.J. (eds.) CG 2005. LNCS, vol. 4250, pp. 120–125. Springer, Heidelberg (2006)
10. Cazenave, T.: Sequential halving applied to trees. IEEE Trans. Comput. Intell. AI Games **7**(1), 102–105 (2015)
11. Ciancarini, P., Favini, G.: Monte Carlo tree search in Kriegspiel. AI J. **174**(11), 670–6684 (2010)
12. Coulom, R.: Efficient selectivity and backup operators in Monte-Carlo tree search. In: van den Herik, H.J., Ciancarini, P., Donkers, H.H.L.M.J. (eds.) CG 2006. LNCS, vol. 4630, pp. 72–83. Springer, Heidelberg (2007)
13. Cowling, P., Powley, E., Whitehouse, D.: Information set Monte Carlo tree search. IEEE Trans. Comput. Intell. AI Games **4**(2), 120–143 (2012)
14. Feldman, Z., Domshlak, C.: Simple regret optimization in online planning for Markov decision processes. J. Artif. Intell. Res. (JAIR) **51**, 165–205 (2014)
15. Ginsberg, M.: Gib: Steps toward an expert-level bridge-playing program. In: Dean, T. (ed.) Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI 1999), vol. 1, pp. 584–589. Morgan Kaufmann (1999)
16. Karnin, Z., Koren, T., Somekh, O.: Almost optimal exploration in multi-armed bandits. In: Proceedings of the International Conference on Machine Learning, pp. 1238–1246 (2013)

17. Kocsis, L., Szepesvári, C.: Bandit based Monte-Carlo planning. In: Fürnkranz, J., Scheffer, T., Spiliopoulou, M. (eds.) ECML 2006. LNCS (LNAI), vol. 4212, pp. 282–293. Springer, Heidelberg (2006)
18. Nijssen, J.A.M., Winands, M.H.M.: Monte-Carlo tree search for the hide-and-seek game Scotland Yard. Trans. Comput. Intell. AI Games **4**(4), 282–294 (2012)
19. Pepels, T., Winands, M.H.M., Lanctot, M.: Real-time Monte-Carlo tree search in Ms Pac-Man. IEEE Trans. Comp. Intell. AI Games **6**(3), 245–257 (2014)
20. Pepels, T., Cazenave, T., Winands, M.H.M., Lanctot, M.: Minimizing simple and cumulative regret in Monte-Carlo tree search. In: Cazenave, T., Winands, M.H.M., Björnsson, Y. (eds.) CGW 2014. CCIS, vol. 504, pp. 1–15. Springer, Heidelberg (2014)
21. Powley, E.J., Whitehouse, D., Cowling, P.I.: Monte Carlo tree search with macro-actions and heuristic route planning for the physical travelling salesman problem. In: IEEE Conference on Computational Intelligence and Games, pp. 234–241. IEEE (2012)
22. Rimmel, A., Teytaud, O., Lee, C., Yen, S., Wang, M., Tsai, S.: Current frontiers in computer Go. IEEE Trans. Comput. Intell. AI Games **2**(4), 229–238 (2010)
23. Russell, S., Norvig, P.: Artificial Intelligence: A Modern Approach, 3rd edn. Prentice-Hall Inc., Upper Saddle River (2010)
24. Sheppard, B.: World-championship-caliber Scrabble. Artif. Intell. **134**(1–2), 241–275 (2002)
25. Tolpin, D., Shimony, S.: MCTS based on simple regret. In: Proceedings of the Association for the Advancement Artificial Intelligence, pp. 570–576 (2012)
26. Winands, M.H.M., Björnsson, Y., Saito, J.T.: Monte Carlo tree search in lines of action. IEEE Trans. Comp. Intell. AI Games **2**(4), 239–250 (2010)

http://www.springer.com/978-3-319-39401-5

Computer Games
Fourth Workshop on Computer Games, CGW 2015, and the Fourth Workshop on General Intelligence in Game-Playing Agents, GIGA 2015, Held in Conjunction with the 24th International Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 26-27, 2015, Revised Selected Papers
Cazenave, T.; Winands, M.H.M.; Edelkamp, S.; Schiffel, S.; Thielscher, M.; Togelius, J. (Eds.)
2016, XII, 179 p. 51 illus., Softcover
ISBN: 978-3-319-39401-5