Comparison of Monte Carlo Tree Search Methods in the Imperfect Information Card Game Cribbage

Richard Kelly and David Churchill
Computer Science
Faculty of Science
Memorial University
{*richard.kelly, dchurchill*}@*mun.ca*

***Abstract*— Non-deterministic imperfect information games pose challenges for Artificial Intelligence (AI) design, as compared to AI for perfect information games. Monte Carlo Tree Search (MCTS), an AI technique that uses random sampling of game playouts to build a search tree rather than domain-specific knowledge about how to play a given game, has been used successfully in some perfect information games. MCTS has also been implemented for imperfect information board and card games, using techniques including sampling over many determinizations of a starting game state, and considering which information set each player belongs to. In this paper, we first describe the imperfect information card game Cribbage and the MCTS algorithm. We then describe our implementation of Cribbage for two players and several MCTS and non-MCTS-based AI players. We compare their performance and find that Single-Observer Information Set MCTS performs well in this domain.**

I. Introduction

Games with imperfect information, including many card, board, and video games, present interesting challenges for game AI. The card game Cribbage, a popular game for 2-4 players, is such a game. Some properties of Cribbage, shared by other card games such as Poker, are:

- **Non-Determinism:** Cribbage is a non-deterministic game, meaning that there is randomness in the game. Cards are dealt randomly to each player at the beginning of each hand and a card is drawn from the deck partway through each hand.
- **Imperfect Information:** Cribbage is an imperfect information game, which means that players do not have knowledge of the complete game state. Each player holds a hand of cards which are hidden from the other player until they are played. Cards yet to be dealt from the deck are also hidden from each player.

Several techniques for handling imperfect information and stochastic game elements in MCTS have been used for board and card games such as Skat, Dou Di Zhu, Settlers of Catan, and Phantom Go [1], [2], [3]. Cribbage was chosen to implement MCTS for in this research because it is relatively easy to implement, but designing a hand-crafted AI to play Cribbage optimally is non-trivial. It is possible to make a hand-crafted AI that plays as well as a novice human player, which was a useful characteristic of the game for testing purposes. We are unable to find previous similar work on the game of Cribbage.

In section II we describe the game of Cribbage and the MCTS algorithm. In section III we present our implementation of Cribbage and several AI players. Section IV details the results of experiments with varying parameters of the AI players and playing the AI players against each other. In section V we discuss the performance of the AI players and finally in section VI we present some possibilities for future research and our conclusions.

II. Background

A. Cribbage

Cribbage is a card game for 2-4 players using a standard 52 card deck. For this research, only the 2-player version of the game was considered. Cribbage is played in repeated hands until a player wins by reaching 121 points at any time during a game. A hand consists of several ordered stages: the deal, throwing, the cut, play, and count. The player who deals alternates between hands, and several game elements are affected by the dealer position.

In each hand, players are each dealt six cards, and both discard two cards face down to a form a third hand (the crib), which is scored by the dealer at the end of the hand. A single community card is cut (drawn) face-up from the deck, which forms the fifth card in each of the three hands for the *count* at the end of the hand. In the *play* stage of the hand, players alternate playing cards from their hand face up while keeping a running sum of the ranks of cards played. When no one can play without going over 31, the play restarts from zero with the remaining cards. When all cards are played, players count the points earned by their hands, and the dealer also counts the crib as their second hand. Players score points in both the play and counting stages by getting sums of 15 and 31, pairs, flushes, straights (runs), and some other combinations.

Cribbage, with only 13 cards in play per hand, has a small game tree for a given deal of the cards, as compared to a game like Skat which has three players each holding 10 cards. If Cribbage were to be played with all cards face-up as a perfect information game, each hand would have at most

$$\underbrace{\binom{6}{2} \times \binom{6}{2}}_{\text{throw}} \times \underbrace{4 \times 4 \times 3 \times 3 \times 2 \times 2 \times 1 \times 1}_{\text{play}} = 129,600$$

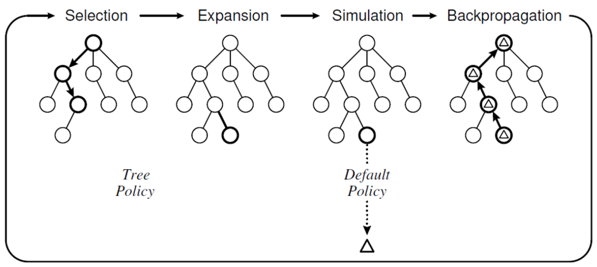

Fig. 1. One iteration of MCTS, from [2].

different ways to be played. However, considering only the cards that one player has seen, there are $9,366,819$ different 6-card hands that the opponent could have at the beginning of a hand. In both cases many of the hands are equivalent. For example, if a player is holding $6\clubsuit$ and $6\heartsuit$ in hand, playing either one of them is equivalent during the play stage of a hand. The imperfect information nature of the game makes the number of possible situations to be considered larger than for a perfect information version.

B. Monte Carlo Tree Search

MCTS was first described in [4]. In MCTS a search tree is built by repeatedly playing out game simulations with random moves and recording the average win rate of different moves. MCTS builds the game tree asymmetrically by focusing on more promising branches. Every node of the search tree in MCTS accumulates information about how successful it has been in previous iterations. That information is then used to bias the selection of child nodes at every level of the search in subsequent search iterations.

MCTS is an *on-line* search–nothing needs to be precomputed to use MCTS–and it is an *anytime* search, meaning it can be stopped whenever a computational or time budget is reached. When the search is stopped, the best move found so far from the root of the tree is selected. The steps for building the MCTS tree are summarized in Figure 1 from [2]. In more detail, the procedure for building the tree is as follows:

- **Nodes:** The tree consists of nodes representing states in the game. Each node keeps track of its visit count and total score or value (win or loss in most games) from visiting that node, as well as a reference to its parent node and child nodes.
- **Selection:** Descend the tree from the root node by following a selection policy until either a terminal node or a node with unexpanded children is reached.
- **Expansion:** If the selected node is not a terminal node, expand it by creating a new node representing an action taken from the parent node and the state arrived at by taking that action.
- **Simulation or Playout:** Play from the expanded node by following a default policy until reaching a terminal game state, which has a value (score for Cribbage) for each player associated with it. The default policy is usually random play but can be otherwise.
- **Backpropagation:** Backup the simulation values to all the nodes visited in the selection and expansion steps.

A key benefit of MCTS is that all that is required to use the algorithm is an implementation of the game that can be used for simulations. Random playouts mean that neither expert knowledge of how to play the game nor evaluations of non-terminal states are needed. Many applications of MCTS do use various enhancements to the algorithm described here, including the use of domain-specific knowledge.

C. Upper Confidence Bound for Trees

The selection algorithm most commonly used for MCTS and used in this research is Upper Confidence Bound or UCB1. It was first applied to MCTS in [5] and called Upper Confidence Bound for Trees (UCT). The UCT algorithm for selecting the next child node v' of a node v to visit is:

$$argmax_{v' \in \ children \ of \ v} \frac{Q(v\prime)}{N(v\prime)} + c\sqrt{\frac{2lnN(v)}{N(v\prime)}}$$

where: $Q(v)$ is the accumulated total value for the player to act at node v from the backpropagation step of previous iterations; $N(v)$ is the number of times v has been visited in the selection step of previous iterations; and c is an exploration constant that can be adjusted for different domains.

Child nodes that have shown little value will be visited when the second term grows, which happens as the parent node is visited. The MCTS algorithm used here visits each child node once before this formula is used to select a child node in future iterations.

III. Methodology

For this research, a Cribbage game and AI players were implemented in Java. A simple graphical interface was made for human play against the AI players. The program implements all the rules found in standard Cribbage play.

Two baseline AI players were implemented for the purposes of testing the MCTS-based players: a random player and a scripted player. The scripted player takes any action which will give the most points immediately after taking that action. In the case of discarding to the crib, it keeps the four cards which on their own, without a fifth card, would have the highest point total at the end-of-hand score count.

In MCTS there are different ways to chose the best move after the computational budget is passed. In this research the child node of the root node with the highest visit count is selected. An alternative method is to choose the child node with the highest average value.

Often, MCTS simulations are played out until the end of a game. The value returned for those simulations is either 1 (win) or 0 (loss). In Cribbage, each player scores a number of points in each hand, and then a new hand is dealt with only the total score and dealer position (which alternates) carrying forward. We only considered playouts until the end of a hand so that more playouts could be done. All MCTS-based agents described in this paper use the difference between the players point gains from the current hand as the value of a simulation.

A. Cheating UCT

The simplest version of a MCTS Cribbage player agent we implemented is a cheating player. The UCT algorithm used in this research is adapted from [2]. The cheating player plays the game as a perfect information game. It has access to all cards, including unrevealed cards in the desk. The Cheating UCT agent ignores the issue of imperfect information, but it is a useful benchmark for other AI players.

B. Determinized UCT

On any players move when there is hidden information in the game state, from that players perspective they may be in any one of many possible game states. That combination of states together form an information set for that player. A determinization of a game state is any state from the players information set. Perfect Information Monte Carlo (PIMC) is used to play games such as the German trick-based card game Skat in [6], [7]. PIMC involves repeatedly taking determinizations from the information set the player to act is in, and then using a standard AI technique for playing out that determinization as a perfect information game. The move chosen at the end of the search is either the one with the highest average value or the one with the most visits over all explored determinizations.

In this research we used determinizations, as in PIMC, but used the Cheating UCT agent to play out each determinization, as opposed to minimax or other methods. The visit counts of all child nodes of the root node are summed across all determinizations, and the action corresponding to the child node with the most visits is returned as best move by the Determinized UCT agent. To create a determinization, each part of the game state which the current player has seen is held fixed, and the rest of the game state is randomized.

In addition to the parameters of execution time and exploration constant, Determinized UCT requires a fixed number of determinizations. Experiments with different numbers of determinizations are presented in Section IV.

C. Single Observer-Information Set MCTS

The algorithm for the final agent implemented in this research, Single Observer-Information Set MCTS (SO-IS MCTS), is adapted from [8]. SO-IS MCTS relies on determinizations, like Determinized UCT, but it builds a single MCTS tree in which nodes correspond to information sets rather than single game states.

A determinization is created before each iteration of the search. As the selection, expansion, and simulation steps are conducted for an iteration, only actions which are compatible with the current determinization are considered. Node selection is based on how good an action has been in previous determinizations that included the same action as a possibility.

In Cribbage, the actions available to the player to act in a given node are the same in all determinizations in which that node is reachable, because that players cards remain the same in all determinizations. The opponent may have certain actions available from a given node in one determinization that are not available in another determinization.

In [8], the selection formula for SO-IS MCTS is modified by considering only the number of times a node has been available for selection, rather than the number of times its parent node has been selected. In normal UCT those numbers are the same, but in SO-IS MCTS, actions that are rarely available would be over-selected if the number of visits to the parent were used in the second term of the formula. The modified formula is:

$$argmax_{\substack{v' \in \, children \ of \ v \\ consistent \ with \ d}} \frac{Q(v\prime)}{N(v\prime)} + c\sqrt{\frac{2lnA(v\prime)}{N(v\prime)}}$$

where: d is the current determinization and $A(v)$ is the number of times the node v has been available for selection.

TABLE I

WIN RATES FOR ALL AI PLAYERS - RANDOM, SCRIPTED, CHEATING UCT ($c = 1.75$, $1s$), DETERMINIZED UCT ($c = 1.75$, $1s$, $d = 100$), SO-IS MCTS ($c = 2.0$, $1s$) (ROW WIN %)

	Random	Scripted	Cheat	Determ	SO-IS
Random	-	2.2	0.0	19.0	1.2
Scripted	97.8	-	24.0	87.5	35.5
Cheat	100.0	76.0	-	97.7	64.0
Determ	81.0	12.5	2.3	-	6.2
SO-IS	99.8	64.5	36.0	93.8	-

IV. RESULTS

Experiments were run to compare the each of the AI players, as well as to compare parameter variations for each of the MCTS-based players. The first dealer has an advantage in Cribbage, since that player benefits from the points available in the crib first. In our tests with random play by both players the dealer won 60% of one million games played. All other tests described in this paper consist of an even number of games with alternating first dealer.

In some experiments the number of search iterations is used as the search budget, while other experiments used time per move as the search budget. All experiments were performed on a machine with an Intel i5 7500 CPU running at 3.4 GHz. For comparison, one second of SO-IS MCTS search used between 50,000 and 150,000 search iterations, which is largely dependent on what stage of a Cribbage hand the search is starting from (a search iteration of a shallower tree is generally faster).

In Table I, the win rates of each AI player against each other are given for a sample of 400 games of each pairing. All other players outperform the random player, and Cheating UCT outperforms all other players. Of the MCTS-based players, Determinized UCT performed worst, winning 81% of games against the random player and 12.5% of games against the scripted player. SO-IS MCTS outperformed all other

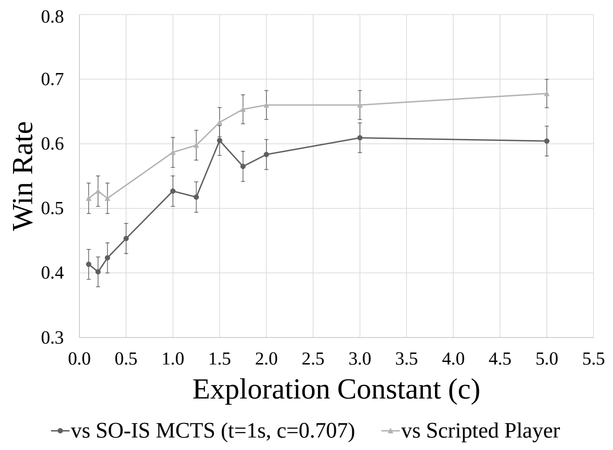

Fig. 2. Effect of exploration constant on SO-IS MCTS ($1s$) performance in 450 games for each value of c.

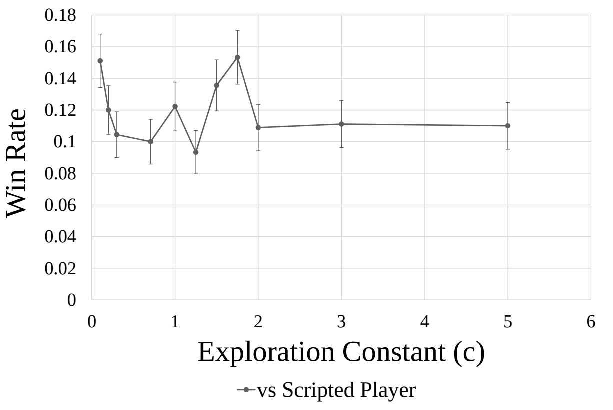

Fig. 3. Effect of exploration constant on Determinized UCT ($1s$, $d = 500$) performance in 450 games for each value of c.

players except the Cheating UCT player, winning 99.8% of games against the random player, 64.5% of games against the scripted player, and 93.8% of games against the Determinized UCT player.

A. Variation of Exploration Constant

The exploration constant, c, is often set to $1/\sqrt{2} \approx 0.707$ in other applications [2]. Therefore, we used $c = 0.707$ as a starting value when first performing tests. In Figure 2, we see that SO-IS MCTS performance improves with higher values of c up to a value of about 2.0. For the main AI comparison tests a value of $c = 2.0$ was used for SO-IS MCTS.

Figure 3 shows the results of varying the exploration constant for Determinized UCT. Performance is highest between 1.5 and 2.0, but the results for very small values of c are also high. That may be because each determinization gives a relatively small game tree to explore, causing the exploration constant to be less important. For the main AI comparison tests a value of $c = 1.75$ was used for Determinized UCT.

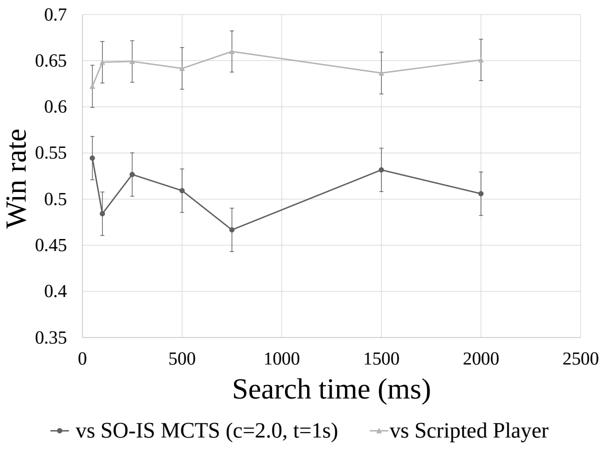

Fig. 4. Effect of search time on SO-IS MCTS ($c = 2.0$) performance in 450 games for each value of t.

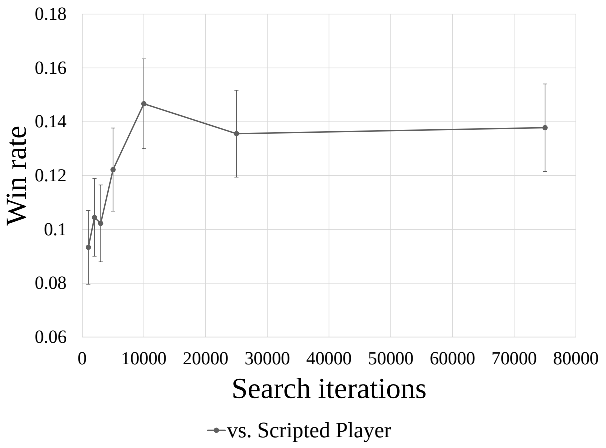

Fig. 5. Effect of search iteration budget on Determinized UCT (k iterations, k/100 determinizations) performance in 450 games at each value of k.

B. Variation of Search Iterations

In Figure 4, we see that SO-IS MCTS performance improves little beyond 100-250ms per move in test of 450 games at each amount of search time against the scripted player. At both 100ms and 2000ms the SO-IS MCTS player wins 65% of games against the scripted player. Against a SO-IS MCTS player set at 1000ms searches, there is no clear trend, and the 50ms player wins 54% of games. In Figure 5 we see that the Determinized UCT player improves in performance as the number of search iterations increases from 1,000 to 10,000, winning 9.3% and 14.7% of games against the scripted player, respectively. Above 10,000 search iterations the Determinized UCT player does not improve for the values that we tested. For the search budgets tested, a higher budget improves performance for both Determinized UCT and SO-IS MCTS, but in each case there is a plateau of performance improvement.

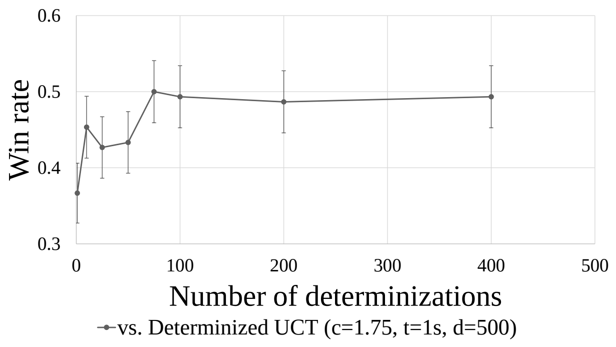

Fig. 6. Effect of number of determinizations on Determinized UCT ($c = 1.75$, $1s$) performance in 150 games for each value of d.

C. Variation of Number of Determinizations

In Figure 6, we see the effect of varying the number of determinizations for Determinized UCT on performance against Determinized UCT with 500 determinizations. Both AI players searched for 1000ms in all tests, with exploration constant $c = 1.75$. Setting the number of determinizations lower than 75 resulted in worse performance than the 500 determinizations player, ranging from 36% to 43% in 150 games. Setting the number of determinizations to 75 or higher (up to 400) resulted in a win rate of about 50% against the player playing with 500 determinizations.

V. Discussion

The poor performance of the Determinized UCT player was surprising. Intuitively, it makes sense that a move that is good in many determinizations would be a good move to take on average. However, playing a given determinization as if it were a perfect information game means assuming both players have access to information that they do not have. It is possible that using MCTS for the playouts in the determinizations is ineffective, since its hard to choose the computational or time budget to allocate to each determinization.

Strategy fusion is an effect in which the searching agent assigns incorrect values to nodes in the tree, because it searches as though it can distinguish between different states in an information set and make a choice based on which state it is in. Strategy fusion is noted as a problem for both Determinized UCT and SO-IS MCTS in [8]. The Determinized UCT Cribbage player always plays as if it can choose actions based on unseen opponent cards. Strategy fusion and other errors of this nature may account for why Determinized UCT performs so poorly for Cribbage.

One consequence of only considering information sets from the maximizing players perspective in SO-IS MCTS is that information about that players hand leaks to the model of the opponent represented by opponent action nodes in the search tree. The acting players cards are held constant in all determinizations, so the nodes of the tree where the opponent is to act will converge to optimal play against the the other players actual hand.

VI. Conclusion & Future Work

In this research, we have shown that SO-IS MCTS outperforms Determinized UCT and a basic scripted player in Cribbage with no game-specific enhancements. Increased search time or variation of other parameters which we tested did not improve the performance of Determinized UCT to the level of SO-IS MCTS.

There are several enhancements that could be made to the algorithms already implemented that would likely offer some improvement at little cost. Stopping MCTS simulations at the end of hands should lead to poor strategy in the final hands of a game. Cribbage games end as soon as any player reaches 121 points, which can happen mid-hand, so the optimal strategy near the end of a game is often to prioritize scoring points earlier in a hand, or to limit opponent scoring. A simple enhancement would be to run simulations until the end of a game whenever a player's score exceeds some threshold.

In [9], measurable properties of game trees that can be used as predictors of the success of Perfect Information Monte Carlo (PIMC) Search are proposed. Future work on Cribbage AI could involve analyzing the properties of its game trees to explain the performance of the Determinized UCT player.

Enhancements to SO-IS MCTS, including Multi Observer-IS MCTS (MO-IS MCTS), which builds a separate tree for each player in a game, are described in [8]. Each node in a tree in MO-IS MCTS corresponds to an information set for that player. This algorithm would address strategy fusion and information leak issues. Implementing the MO-IS MCTS algorithm for Cribbage and for more complicated games with larger search trees is an area for potential future work.

References

[1] J. Schäfer, M. Buro, and K. Hartmann, "The uct algorithm applied to games with imperfect information," *Diploma, Otto-Von-Guericke Univ. Magdeburg, Magdeburg, Germany*, 2008.

[2] 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 Transactions on Computational Intelligence and AI in games*, vol. 4, no. 1, pp. 1–43, 2012.

[3] I. Szita, G. Chaslot, and P. Spronck, "Monte-carlo tree search in settlers of catan." *ACG*, vol. 6048, pp. 21–32, 2009.

[4] R. Coulom, "Efficient selectivity and backup operators in monte-carlo tree search," in *International conference on computers and games*. Springer, 2006, pp. 72–83.

[5] L. Kocsis and C. Szepesvári, "Bandit based monte-carlo planning," in *ECML*, vol. 6. Springer, 2006, pp. 282–293.

[6] M. Buro, J. R. Long, T. Furtak, and N. R. Sturtevant, "Improving state evaluation, inference, and search in trick-based card games." in *IJCAI*, 2009, pp. 1407–1413.

[7] T. Furtak and M. Buro, "Recursive monte carlo search for imperfect information games," in *Computational Intelligence in Games (CIG), 2013 IEEE Conference on*. IEEE, 2013, pp. 1–8.

[8] P. I. Cowling, E. J. Powley, and D. Whitehouse, "Information set monte carlo tree search," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 4, no. 2, pp. 120–143, 2012.

[9] J. R. Long, N. R. Sturtevant, M. Buro, and T. Furtak, "Understanding the success of perfect information monte carlo sampling in game tree search." in *AAAI*, 2010.