Kougentakos Alexandros

Evaluating the Performance of Determinized Monte Carlo Tree Search (MCTS) in Team-Based, Turn-Based Card Games: A Case Study on "Tichu"

Supervisor: Hoefman Kevin

Coach: Van der Kelen Cedric

Graduation Work 2024-2025

Digital Arts and Entertainment

Howest.be

CONTENTS

ABSTRACT

This study aims to evaluate whether the Determinized Monte Carlo Tree Search (DMCTS) is viable for use in Turn-Based and Team-Based games with imperfect information. The research focuses on benchmarking the performance of DMCTS against a cheating MCTS agent and random agents.

To quantify the algorithm's playing performance, experienced human players were asked to play against the AIs and complete a questionnaire on their performance.

The study revealed several findings about DMCTS's performance in Tichu. First, increasing only the iterations yields better results than increasing only the determinizations. Secondly, in order to increase the determinizations, you need to have a sufficient number of iterations to follow them.

Another interesting discovery is that the algorithm's computation time per move decreased linearly throughout the game. This presents an opportunity for dynamic iteration adjustments during the game. This is unique since not many card games have a constantly decreasing pool of cards. It usually fluctuates.

Finally, this study showcases the importance of using some method to compare an algorithm to something that will give it credibility. While on-paper the algorithm seemed to perform very good, when playing against human players it revealed several weaknesses and not the expected performance.

This study contributes to the lacking but growing field of research in AI for imperfect information games. It also fills the gap for team-based limited information games and proves that DMCTS is a potential candidate for such games.

KEYWORDS

Monte Carlo Tree Search, MCTS, Determinization, DMCTS, Artificial Intelligence, AI, Card Game, Tichu, Bot, Imperfect Information Game, Game AI

PREFACE

Artificial intelligence has always fascinated me. The idea of crafting algorithms and neatly integrating techniques to surpass human expertise in complex tasks is nothing short of inspiring. My fascination began with the story of Deep Blue, the computer that famously defeated the greatest chess champion at their own game.

Tichu is a game I have cherished and played with friends for years. Though far from a master of the game, I was eager to leverage my technical skills to create an AI capable of surpassing my own abilities. While I knew this challenge would be anything but simple, it represented a step forward in combining my passion for AI with a game I love.

LIST OF FIGURES

INTRODUCTION

In recent years, artificial intelligence has become much better at beating expert humans in increasingly difficult games. Following the success of such AIs in perfect information games like chess and Go (Silver et al., 2017), there has been a shift in interest towards imperfect information games such as poker (Moravčík et al., 2017). However, this area is still lacking compared to the previous one (Whitehouse et al., 2011).

Those games, however, benefit from centuries of human expertise and research, which made it easier for researchers to use reinforcement learning algorithms with the help of evaluation functions. Tichu does not offer such extensive documentation for quality playing, so DMCTS offers a promising alternative that requires no domain-specific knowledge, with the exception of a reward function. A reward function only needs to judge the outcome of a game, which is considerably easier to do. Simply give a higher value to moves that ended up in a win and a lower value to moves that ended up in a loss.

This study investigates the application of DMCTS in Tichu, a complex team-based card game characterized by imperfect information and a large branching factor. This research paper aims to evaluate DMCTS's effectiveness in handling the game's unique nature while maintaining computational efficiency.

Note, to make a clear distinction between computational performance and playing performance, I will be referring to them as such from now on. Playing performance simply refers to the in-game performance.

Tichu is a complicated game with many stages and edge-case rules. I did my best to incorporate all of these into the AI's capabilities to match what it would look like during a game closely. However, certain adjustments had to be made to study the algorithm's decision-making capabilities more closely. This is discussed in more detail in the Rule Modifications section.

LITERATURE STUDY / THEORETICAL FRAMEWORK

TICHU

Tichu is a trick-taking shedding card game originally from China. It is also known as Tai Pan. Tichu uses a unique deck consisting of 52 cards with four suits (2 through Ace) and four additional "special" cards: the hounds, the dragon, the phoenix, and the mahjong.

GAME LOOP

In Tichu, players take turns and throw card combinations down. You can only throw the same type of combination that is already down. You can choose from all the possible combinations if there are none. After there is only one person with cards left, the round ends, and the points are counted. There are exceptions to this; for example, if two players from the same team go out before the other team, the game stops there, and the team that's out gets awarded 200 points. This is called a 1-2. There are more rules regarding the point distribution when people go out, but they are not crucial for this paper.

POINTS

There are only a few cards that are worth points. 5's are worth 5 points, 10's, and Kings are worth 10 points.

The dragon and the phoenix are also worth points, and they are mentioned in a subsequent section. The total points of each round add up to 100.

COMBINATIONS

- **Singles:** Just a single card
- **Pairs:** A pair of two cards of the same value
- **Triplets:** A triplet of 3 cards of the same value
- **Steps:** Multiple doubles of increasing value by 1. E.g., 2,2,3,3. These can keep going as long as you want
- **Straight:** 5 or more cards in a row. To beat this, you also need to match the number of cards
- **Full House:** A pair and a triplet. To beat this, you only need to throw a full house with a greater triplet.
- **Four-of-a-kind bomb:** Four-of-a-kind is a bomb and can beat any other combination in the game. The bombs can also be beaten by better bombs.
- **Straight Bomb:** 5 or more cards in a row of the **same color.** This bomb is greater than the four-of-a-kind bomb.

Tichu offers a unique mechanic to bombs; They can be played **outside of your turn**. If you can beat the previous combination **(THERE MUST BE ONE),** you can throw it at any point.

SPECIAL CARDS

The special cards are unique, and all offer a different power:

- **Dragon:** The dragon is the biggest single card there is. You can only use it as a single, but it beats every other single. This card is worth **25 points**, but any trick WON by the dragon must be given to an opponent.
- **Phoenix:** The phoenix acts as a joker. It can be made into any non-special card. If you have a 2,3,5,6, you can use the phoenix as a 4 to form a straight. If played as a single, it counts as 0.5 more than the card on top of which it was played. You **cannot** play it on top of a dragon card. This card is worth **-25 points.**
- **Mahjong:** Whoever has the mahjong gets to start the game and can also make a "wish." A wish consists of 1 card, and if the player is legally allowed to play that card, they are forced to do so. The wish remains for the entire round until it is fulfilled. Mahjong can be played as a single card with a value of 1 but can also be played as part of a straight, again as a 1.
- **Hounds:** The hounds can only be played when it's your turn, and there are no other cards on the table. All they do is pass your turn to your teammate. If your teammate is out, they go to the next person.

Figure 1 - The special cards in order

DECLARATIONS

At the beginning of the game, players are dealt 8 cards and asked if they would like to declare Grand Tichu. This declaration means that the player thinks they will go out first. If they do, their team will gain an additional 200 points. If they don't, they will lose 200 points.

There is a smaller variant of this called just **Tichu**, and you can declare this at any point as long as you have not played your first card. This declaration is the same but is only worth 100 points.

CARD TRADING

When the players first receive their 14 cards they must trade 3 cards. They need to give/receive a card to/from every other player.

RULE MODIFICATIONS

NO TICHU/GRAND TICHU DECLARATIONS

This research focuses on the application of DMCTS in the core game loop of Tichu. Trying to make MCTS work for both phases and consider Tichu/Grand Tichu calls would be unnecessary since some other heuristic algorithm would probably handle them. While Tichu calls are an essential part of the game, if the algorithm can perform well without them, there is not much reason to believe that it would perform worse with them.

NO MAHJONG WISHES

This is possible to implement in the current version of the algorithm without modifying the algorithm, however for every move containing the mahjong you would need to add 14 more moves. Each move with a different wish would represent a different game state.

In an average situation this would double or even triple the possible moves. But if you had a straight of 7 cards starting at the mahjong that would be 3 different straights containing the mahjong which are 3 different combinations. That means you would have 42 extra nodes for the algorithm to check.

This would unnecessarily use up computational power.

NO ASYNCHRONOUS BOMBS

This would require major algorithm overhauls since it doesn't follow the traditional turn-based trick-taking ruleset and would be a considerable challenge on its own. The game environment I made also did not support this, and it would be outside the scope of this research paper.

MONTE CARLO TREE SEARCH IN GAME AI

When thinking of an algorithm that you can use in turn-based games, the simplest one that comes to mind is Minimax. Minimax is a very simple but strong algorithm that guarantees the best possible move in a certain situation. The problem with Minimax is its' branching factor.

For a simple game like Tic-Tac-Toe, the highest number of possible moves the algorithm would need to look at is 9. This means that there are 255.168 different boards in Tic-Tac-Toe ("How many Tic-Tac-Toe (noughts and crosses) games?," n.d.). Any modern computer can calculate the whole tree relatively fast.

In contrast, chess has a considerably higher number of possible moves, with the lower bound estimated at 10^{120} ("Shannon number," 2025). To compensate for astronomical branching factors like these, experts have devised something called an evaluation function. While evaluation functions are valuable, they rely heavily on human expertise. Chess, with its centuries of documentation, is well-suited for such functions. However, for less-studied games like Tichu, developing an effective evaluation function becomes considerably more challenging.

This is where Monte Carlo Tree Search (MCTS) proves to be particularly useful. MCTS has established itself as a significant algorithm in turn-based games, having been applied to several different games (Browne et al., 2012) (Świechowski et al., 2023).

HOW IT WORKS

The fundamental concept of MCTS is relatively straightforward. When the algorithm is queried for a move, it is provided with the current "state" of the game. This “state” varies depending on the game, but it usually consists of relevant information such as the cards played previously, the current cards, the score, etc.

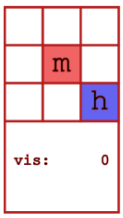

Figure 2 - MCTS Base node for Tic-Tac-Toe

Figure 2 (“MCTS Tic-Tac-Toe Visualization,” n.d.) shows a base node from the MCTS algorithm representing the game's current state. This is the state when you request a move from the algorithm.

From there, the algorithm will choose one of the possible moves it can make and continue to the next phase. The move selection is pseudo-random at first because of the small sample size, but later, there is an exploitation vs exploration dilemma. The standard implementation uses the UCB1 formula (Kocsis and Szepesvári, 2006).

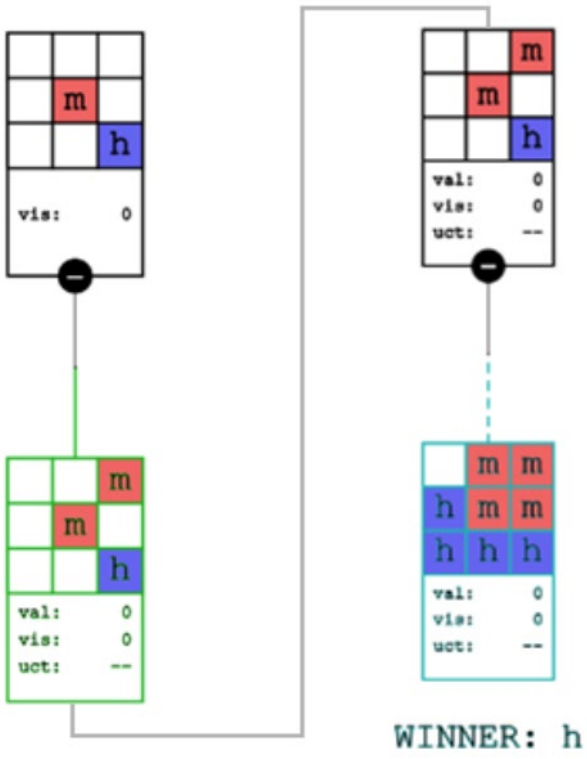

Figure 3 - Move selection and game simulation

Figure 3 shows the random move that the algorithm chose to play. From there on, it will randomly simulate a game. This part is entirely random. Figure 3 also showcases the result of this match. In this instance, the game ends with a win for the human player, meaning this move will now get a negative score since this tree is formed from the perspective of the machine. This gets repeated multiple times. The number of times this gets repeated is referred to as the iteration count.

LIMITED INFORMATION

Currently, the AI is being given information it should not have: the other players' cards. MCTS gets modified to play the game fairly with limited information in two main ways. The Information Set MCTS and the Determinized MCTS. I decided to test the Determinized MCTS as while it suffers from some playing quality performance issues, it is better suited to handle the large branching of Tichu (Whitehouse et al., 2011).

DETERMINIZED MCTS

Determinized Monte Carlo Tree Search (MCTS) is an adaptation of the MCTS algorithm designed to handle games with imperfect information, such as card games where players' hands are hidden from each other. The key idea behind determinization is to create multiple "determinized" versions of the game state, where the hidden information is randomly filled in, and then apply standard MCTS to these determinized states (Bax, n.d.; Kelly and Churchill, n.d.).

The process of the determinized MCTS can be broken down into the following steps:

- **Determinization**: When the algorithm is called to make a move, it creates multiple instances of the current game state, randomly assigning values to the unknown information. E.g., the opponent's cards (Bax, n.d.).
- **MCTS on Determinized States**: For each determinized state, the algorithm performs a standard MCTS search, building a separate search tree for each determinization (Kelly and Churchill, n.d.).
- **Action Selection**: After running MCTS on all determinizations, the algorithm aggregates the results to choose the best action. This is typically done by selecting the action that performed best on average across all determinizations (Whitehouse et al., 2011).

Essentially, we just added one more stage to our previous four.

Determinized MCTS has several advantages that make it suitable for games with large branching factors like Tichu:

- **Computational Efficiency:** It can handle games with larger branching factors more efficiently than Information Set MCTS (ISMCTS), as it reduces the effective branching factor in each determinization (Lanctot et al., 2014).
- **Parallelization:** The algorithm is naturally parallelizable, as each determinization can be processed independently (Chaslot et al., 2008).

Furthermore, it was also very easy to implement into the MCTS algorithm, which I had to write in the first place to create the cheating agent.

However, determinized MCTS also has some limitations:

- **Strategy Fusion:** The algorithm may make suboptimal decisions by assuming it can make different choices in indistinguishable states across different determinizations (Kelly and Churchill, n.d.)
- **Sampling Error:** If either the number of iterations or determinizations is low, the algorithm may make poor decisions due to insufficient sampling of the possible game states (Bax, n.d.).

PARALLELIZATION

MCTS follows the principle of Single Instruction Multiple Data very well, so it's naturally very easy to parallelize, at least the "core" loop, which is the most computationally expensive. The most efficient way to parallelize this algorithm is to split the core loop of the four steps (Selection, Expansion, Simulation & Backpropagation) into a function that runs for a number of iterations, and then evenly distribute the work onto different threads (Chaslot et al., 2008).

DETERMINIZED PARALLELIZATION

This part is a little trickier. To effectively parallelize the DMCTS algorithm, you need to parallelize two things. The iterations and the determinizations. The issue is that the determinizations happen within the iterations. Trying to parallelize **both** will lead to nested parallelization. This is quite tricky to handle without a more robust system like a thread pool.

The other solution is to choose one of the two to parallelize, either the iterations or the determinizations. The question is, which one?

The number of iterations per determinization of Monte Carlo usually follows a large ratio like 1/100 (Kelly and Churchill, n.d.), so it would make sense to multithread the one with the larger number.

SMALL OPTIMIZATIONS

This paper explores the raw performance of DMCTS without utilizing move pruning or other optimization techniques. Nevertheless, several straightforward measures were taken to improve the algorithm's speed. The first of these measures, which likely contributed the most to performance enhancement, is the implementation of multithreading in the algorithm.

Another small change that was made was that the algorithm does not consider color in its move generation. For example, if you had 3 twos, a blue, a red, and a green one, this allows you to create 6 (3!) different pairs. The algorithm only considers this as 1 possible pair of twos. This significantly reduces the number of possible moves that the algorithm must go through. As noted by the authors of a related study, "*The discrete action space of Tichu is huge with approximately 10^9 different valid combinations that can be formed from 14 cards. We can reduce the action space to approximately 10^3 by ignoring the colours of the cards*." (Avarikioti et al., n.d.)

A side effect of this is that the algorithm doesn't detect straight bombs. You can still make modifications to the algorithm and have it especially consider the color of a bomb after it has been found. But once again, those are changes that should be checked after the validity of the base algorithm has been established.

RESEARCH

1. ITERATION COUNT

The first thing I needed to do was check whether the algorithm was performing correctly. To do this, I created four agents: two that use an iteration count of 1, essentially a completely random move, and another two with an increasing number of iterations. I did this to see the effects of increasing the number of iterations and if it yielded an increase in playing performance as well as its' effects on computational performance.

All the following measurements were taken using a custom game repetition system. Essentially, I made a small class that collected the results for each AI's actions. After **100 rounds,** it would perform some calculations, such as getting the average move time, and then output everything into a file. The game would automatically reset once they reached 500 points. This worked out well since the Ais are programmed to only consider the current round.

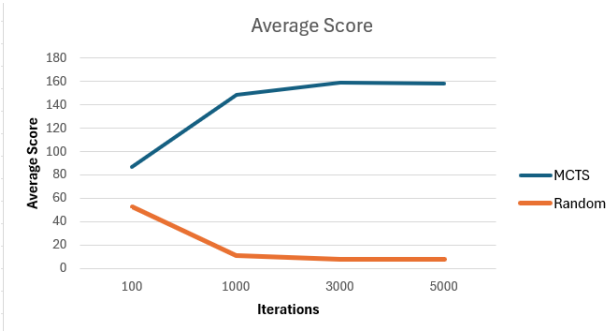

Figure 4 - Average score as iterations increase

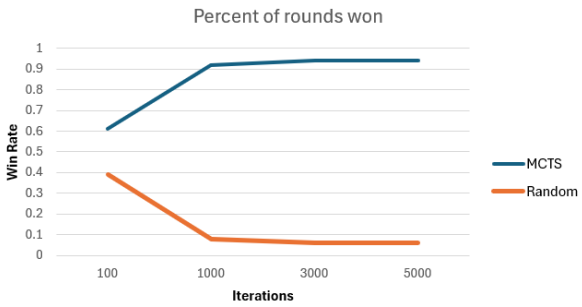

Figure 5 - Percent of rounds won as iterations increase

As we can see from Figure 4 and Figure 5, there is a clear indication that increasing the number of iterations also increases the playing performance of the AI. It also shows that somewhere around the 3000-5000 iteration mark is where the performance benefits stop. We cannot, however, be certain of the second extrapolation, as there is a maximum score that the AI can achieve. Additionally, failing to get above 90% of that score every round against random agents does not mean that this is the flat line. It could be that you simply cannot consistently get that many points. Each game of Tichu lasts up to 500 points, and as expected, there is a very close correlation between the average score gained and the games won, as we can see in Figure 6 when compared to Figure 4.

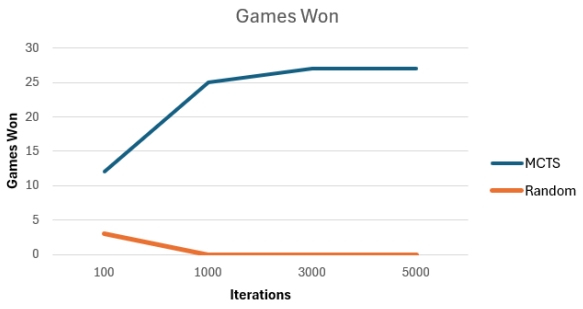

Figure 6 - Game won as iterations increase

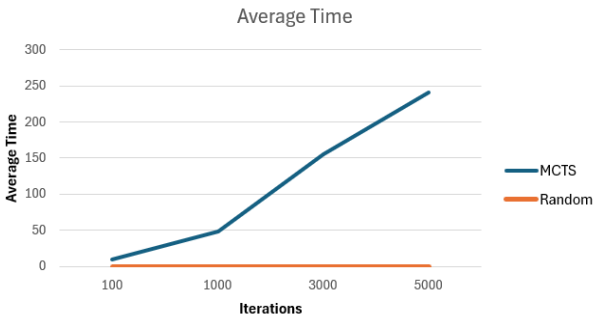

Figure 7 - Average time per move as iterations increase

Finally, as expected, the time it takes to calculate a move also increases, pretty much linearly, as the number of iterations increases as well. This is clearly visible in Figure 7. Note that the timings on the Y-axis are in milliseconds.

2. DETERMINIZED MCTS ITERATION COUNT

After implementing the DMCTS algorithm, I also had to perform the same test to make sure that it was functioning as expected. For the following tests, I kept the determinization count at 10 and only changed the iterations.

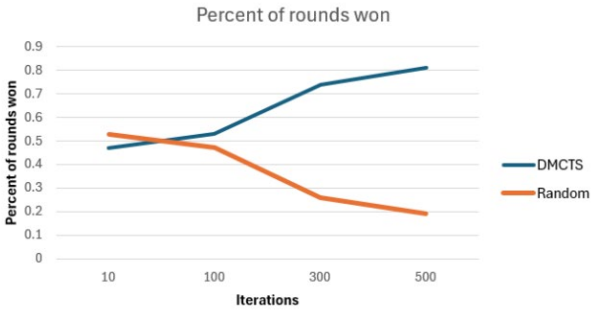

Figure 8 - Percent of rounds won through the game

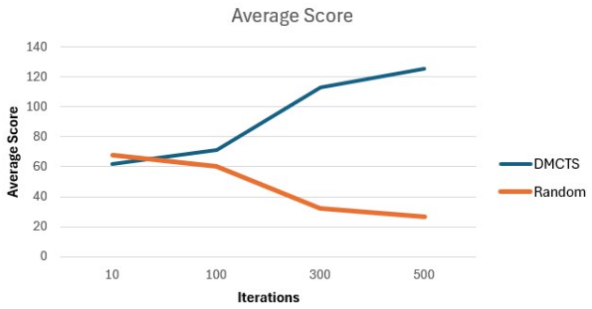

Figure 9 - Average score through the game

Note that the number of iterations on the X-axis appears to be smaller than the one for the cheating MCTS algorithm, but this algorithm also uses determinizations. Since that value is fixed at 10, that means that for each iteration, it performs 10 determinizations, which are essentially just the MCTS algorithm with random cards. So, the total number of iterations is 10 times the amount of the X-axis.

Interestingly enough, the algorithm seemed to perform worse than the random with a very low iteration count. I assume this is because the reward function favors 1-2s so the AI might just be trying to reach illogical moves that stumbled upon a 1-2 at some point. But we can see that it quickly picks up a higher win rate.

Another thing we can see from Figures 8 and 9 is that the DMCTS does not reach the flat line as early as the cheating MCTS. Increasing the iterations higher is necessary to reach that point.

3. DETERMINIZATION COUNT

DMCTS differs from the traditional MCTS in that it has two values you must manage. One is the iteration count, which, as we can see, functions similarly to that of MCTS; increasing the iterations also increases the playing performance. However, increasing the determinizations does not necessarily increase the playing performance.

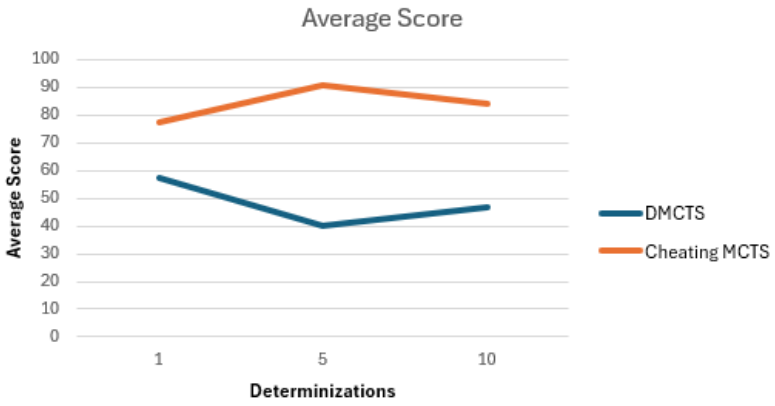

Figure 10 - Average score with different determinizations

As we can see from Figure 10, increasing the number of determinizations had a negative effect on the algorithm's playing performance (Cowling et al., 2012). This is probably because of sampling error, one of the downsides of DMCTS. The easiest way to fix it is to increase the number of iterations relative to the number of determinizations.

Metric	Cheating MCTS	Determinized MCTS
Final Scores:	6	9
Round Win Percentage:	49%	51%
Average Points:	61.25	68.75

Figure 11 - Average score with different determinizations

As we can see from Figure 11, increasing both the iterations and the determinizations with a large ratio seems to perform better. Note that the cheating MCTS used 3,000 iterations in this test, while the Determinized MCTS used 10,000 with 15 determinizations. This is roughly 3 times the iteration budget, but this was done to mitigate the difference in information. Typically, you would use the same number or a little larger when comparing a cheating and a non-cheating MCTS variant, but since Tichu is a game that lasts for a lot more moves than a typical game where this algorithm is implemented such as hearts (Cowling et al., 2012) or Scopone (Di Palma and Lanzi, 2018), I allowed a larger gap to make up for the depth of the tree search.

When comparing this to a DMCTS AI, in Figure 12, using only 1 determinization with 10,000 iterations, we can see that the one with 15 determinizations managed to perform slightly better. This proves the theory that higher determinization counts also require many more iterations to go with them; otherwise, the randomness creates too much noise.

Metric	Cheating MCTS	Determinized MCTS
Final Scores:	11	4
Round Win Percentage:	58%	42%
Average Points:	73.15	57.85

Figure 12 - Average score with less determinizations

The number of iterations seems to be game-specific and is probably tied to the branching factor of the game. The results of this simulation suggest that increasing the determinization count yields better results, but only if sufficient iterations are provided.

4. TIME TO MOVE THROUGH THE GAME

While capturing the data needed for this research paper, I noticed that the AI would tend to speed up towards the end of the game. I wanted to verify this, so I rewrote the data capture class to support it.

The problem with presenting move timings is that certain games last for more moves than others. To address this problem, I normalized the moves into a 10-move range.

4.1 NORMALIZATION PROCESS

Each game consists of a variable number of moves (e.g., 18 in one game and 12 in the next). We will normalize the move indices by scaling them to a standard range.

$$Normalized\ Move\ Index\ = \frac{Move\ Index}{Total\ Moves\ in\ Game}$$

For example:

- **Game 1 (*18 moves*):** Move 1 corresponds to 1/18, Move 9 corresponds to 9/18 =0.5 etc.
- **Game 2 (*12 moves*):** Move 1 corresponds to 1/12, Move 6 corresponds to 6/12=0.5 etc.

We then divide the range into 10 segments from 0 to 1:

$$[0, 0.1), [0.1, 0.2], \ldots . [0.9, 1.0].$$

We then assign each move's timing data to the corresponding interval. For instance:

- **Game 1: Move 1 (*0.056*)** belongs to [0,0.1) and Move 9 (0.5) belongs to [0.5,0.6).
- **Game 2: Move 1 (*0.083*)** belongs to [0,0.1) and Move 6 (0.5) belongs to [0.5,0.6).

4.2 RESULTS

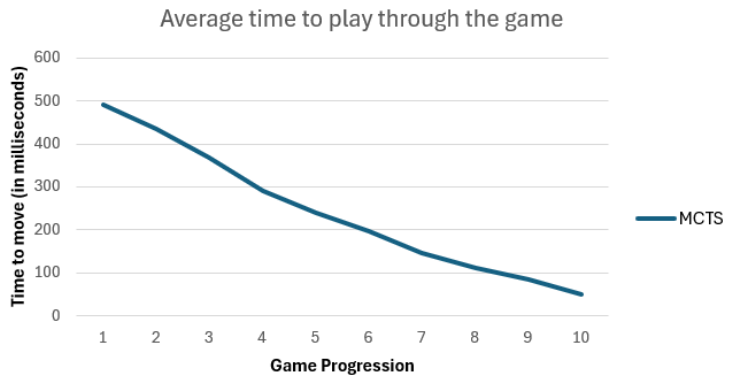

Figure 13 - Time to move through the game

As we can see from Figure 13, the AI linearly decreases the time needed to calculate a move. Initially, I did not understand why this was. Still, after some research, I realized that as the game progresses, when you query the algorithm for a move, you essentially start a lot deeper in the search tree than you did at the start of the game. As a result, reaching a terminal state takes significantly less time.

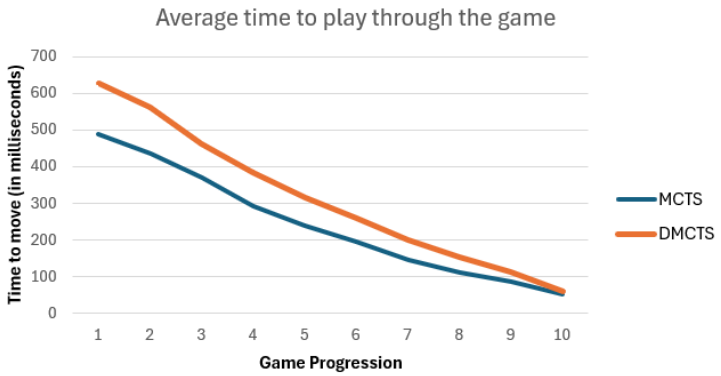

Figure 14 - Time to move through the game comparison at 5.000 iterations

We can see the same trend from the Determinized algorithm in Figure 14. Both of those algorithms are running at 5.000 total iterations each. As expected, the Determinization adds some overhead, but the difference is not massive.

This opens many possibilities. The middle-to-end game of Tichu is more critical as it determines who goes out first and puts pressure on the other teams to prevent their opponents from doing so. Whoever goes out first has a considerable advantage as the points collected by the player who did not go out are given to them.

You could, for example, implement an iteration counter that is not fixed but increases opposite in relation to the number of cards left in the game.

5. HUMAN EXPERT ANALYSIS

Following the results of the previous research, I decided to distribute the game with all the AIs using the DMCTS algorithm with 15,000 iterations and 5 determinizations for half of them, 15,000 iterations, and 10 determinizations for the other half.

Players were asked to play two full games (roughly 10-12 rounds), fill out a questionnaire and hand in the data that the software automatically generated. It should be mentioned that when rating, participants were asked not to use the number 7, except for the final question. This is because the number 7 is often considered a "safer" option for participants to choose and tend to go for that more often. Removing this option would force them to consider their option further before deciding.

THE MODELS FACE OFF

When comparing the two models given to the players in teams, the one with more determinizations, as expected from the previous results, outperformed the one with less. You can see those results in detail in Figure 15.

Metric	15,000 It + 5 Det	15,000 It + 10 Det
Final Scores:	7	10
Round Win Percentage:	45%	55%
Average Points:	62.25	73.75

Figure 15 – Table comparing the distributed agents

QUESTIONNAIRE RESULTS

Agent	15,000 It + 5 Det	15,000 It + 10 Det	Combined
Overall Rating	6.3	7.8	7
Consistency	6.2	7.8	7
Cooperation	3.6	5.1	4.4
Special Card Usage	3.5	6.5	5
Point Collecting	7.5	8	7.75
Compared to player	4.8	5.3	5

Figure 16 - A table comparing the averaged ratings from the questionnaire

As we can see from Figure 16, the players consistently rated the algorithm using more determinizations higher across all metrics. They also agreed that the overall performance was decent. We especially notice a significant gap in overall performance between the two versions.

However, 10 out of the 12 participants stated that the AI was worse than them. So, while the algorithm performs decently, humans still think they often perform better.

The following charts take a more in-depth look at the range of answers I received from the questionnaire. The rating questions that were asked are as follows:

Q1: *How would you rate the AI's overall ability to play Tichu effectively?*
Q2: *How would you rate the AI's consistency across multiple rounds?*
Q3: *How would you rate the AI's ability to cooperate with a teammate?*
Q4: *How would you rate the AI's ability to use special cards effectively?*
Q5: *How would you rate the AI's point-collecting ability? Did it actively attempt to gather points?*
Q6: *Assuming YOUR skill level is a 7/10, how would you rate the AI's ability to play compared to you? 7 Being basically equal.*

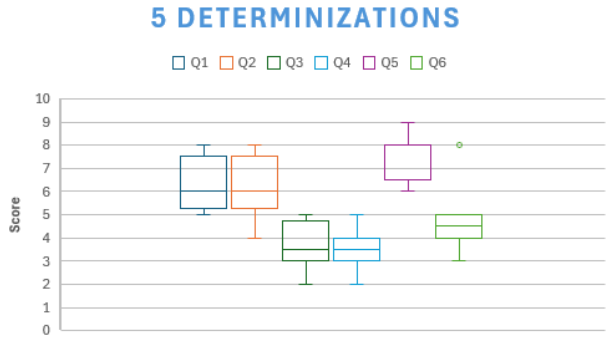

Figure 17 - Answer distribution for A.I. agent with 5 determinizations

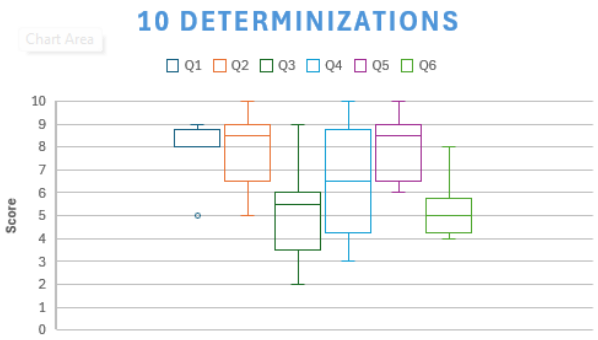

Figure 18 - Answer distribution for A.I. agent with 10 determinizations

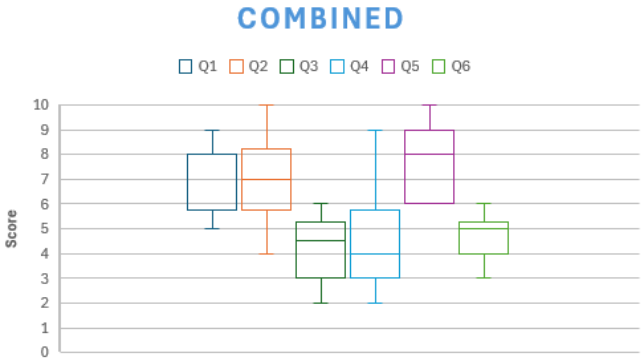

Figure 19 - Answer distribution of both models

The candles on either side of the bars represent the outliers. They indicate the full range of ratings the AI agents received on this question. The bars represent the more common range of ratings.

As we can see, the average rating for the overall performance is quite good. Participants rated it between 5 and 9, with the more common range being between 6 and 8. The worst rating, however, is the ability to use unique cards and be a good teammate.

When the participants were asked if they noticed any weird patterns, six of them mentioned that their own teammates would often throw a higher combination on their own. While this is acceptable for lower combinations occasionally, with good reason for higher ones, you generally try not to beat a combination if your teammate throws it down.

Four participants also mentioned that the AI would tend to use the dragon card very badly, often throwing it out on its own. I suspect this has to do with the fact that when beating a combination with the dragon, all the points are given to the other team. Three out of those four were given the 5 determinization agents, and as we can see when comparing Figure 18 to Figure 17 on Q4, the 10 determinization agents used the cards better and more consistently. However, the outlier ratings suggest that they also suffered from this issue, but it was less common.

While the participants were playing the game, some data was also collected that the participants were asked to hand in with their questionnaire. This is the same data that I was collecting for the simulation matches.

Agent	15,000 It + 5 Det		15,000 It + 10 Det	
Team	Human – AI	AI – AI	Human – AI	AI – AI
Average Score	72.85	53.15	87.35	42.65
Round Win%	57%	43%	73%	27%

Figure 20 - Comparison between teams in human games

It should be noted that only 9/12 handed it, so 5/9 were games of the 10 Determinization algorithm, and the other 4 were part of the 5 Determinization algorithm. This might lead to a slight bias toward the former algorithm.

Regardless, as we can see from Figure 20, the team that had a human outperformed AI agents formed the team that had. If we recall the answers the participants gave in their questionnaires, the 10 Determinization model was graded higher in cooperation, which explains this gap.

I would have expected that if two agents had been better, they would have significantly outperformed humans. Still, since Tichu is a demanding team game, it could be possible that if the humans received better support from the 10 determinization agent, they also performed better themselves.

DISCUSSION

This research confirms that the DMCTS algorithm can be a valuable tool in imperfect information games like Tichu, demonstrating its effectiveness in both computational and playing performance. However, some caveats and areas for improvement emerged from the findings.

Another thing that stood out is how the algorithm behaves with special cards, like the Dragon or Phoenix. It's not great at using them effectively. The Dragon, in particular, seemed to confuse it, as it would sometimes throw it out at odd moments. This is probably because the reward function isn't set up to handle these situations appropriately. Perhaps the AI is considering that if the opponents throw a king or a 10 and it throws the dragon, the opponents would get even more points. One idea could be to tweak the reward function to offer a small reward equal to the value of a dragon to the team that goes out first. The AI might be more inclined to use the dragon to go out first or ensure their teammate does.

It is hard to fine-tune specific issues like these in MCTS, as the reward is handed out at the end rather than right after the move. This is an area where reinforcement learning shines.

Regarding the feedback from human players, most agreed that the AI was at a decent enough level. The biggest issue was the cooperation aspect, which is even harder to tackle. A machine will play like a machine unless instructed to mimic human play. But if this style of play were better, we would expect the team with two AI agents to outperform the one with the human. Perhaps having two humans in a team would yield different results.

CONCLUSION

This study investigated whether the DMCTS algorithm could manage a complex team-based game like Tichu. Overall, the results were promising. The algorithm performed reasonably well, even against human players. However, when paired with a human teammate, its performance was lacking and left much to be desired.

Additionally, several other noteworthy observations were made.

ITERATION AND DETERMINIZATION DYNAMICS

The research demonstrated that increasing the iterations improved the AI's performance. Still, the limits of this were not visible, as increasing the opponents' competency seemed to require more iterations for the algorithm to perform better.

However, the determinization count showed a more nuanced performance curve, requiring a substantial number of iterations to reach optimal performance compared to the cheating MCTS algorithm.

TIME TO MOVE DECLINE

A unique observation was the linear decrease in computational time per move as the game progressed. This is not a characteristic of the algorithm but rather of its application in this specific game. The Tichu card count starts high but linearly decreases as the game progresses.

ALGORITHMIC POTENTIAL

This study showcased that even the base version of the DMCTS algorithm performed decently in a team-based setting inside a game with a considerable branching factor. Many optimizations and changes can be applied to the algorithm to boost computational performance, so there is some potential for creating an expert-level player using this algorithm.

FUTURE WORK

While this research has demonstrated the potential of DMCTS in a game like Tichu, there are still plenty of directions to explore and areas to refine. First and foremost, the relationship between determinizations and iterations needs more attention. Finding the ideal balance between the two could improve the algorithm's consistency and overall performance. A more systematic investigation into how this ratio changes depending on the branching factor or game complexity would be valuable.

Once that uncertainty is addressed, moving pruning could be implemented, which offers several advantages. It reduces noise from the randomness in the algorithm by disregarding moves that are unlikely to be successful, saving iterations. This would, in turn, allow us to lower the number of iterations and save on some computational efficiency, which could be used in combinations with other heuristics.

For instance, instead of randomly generating moves in the simulation phase, you could use some heuristic to generate more logical moves. This would further reduce noise and potentially nullify its own overhead if you require fewer iterations.

CRITICAL REFLECTION

Working on this project has been a challenging but rewarding experience that has taught me a lot about both AI and my own approach to problem-solving. When I first started, I underestimated how much complexity a game like Tichu could bring to an algorithm like DMCTS. I honestly assumed that it would have been completed a lot earlier than it was.

A huge lesson was balancing ambition with practicality. I have wanted to work on this for a very long time, and I planned to not only verify my current findings but also add support for all the features of Tichu in this algorithm. However, I realized very early on that cutting those out would allow me to perform much more thorough tests and would yield more meaningful results.

Another major takeaway was iteration. Not in the algorithm, there was enough of that there. Iteration in my workflow. There were many occasions when I wanted to speed through implementing things, but I stayed my course and implemented and thoroughly tested every feature. This saved me a lot of time and gave me peace of mind, knowing for certain that specific features had been tested and verified to be working correctly rather than assuming they were.

While the AI did not perform as well as I had expected, I am glad I took on this research project. It opened my eyes to a whole world of possibilities and interests for me to explore in the future.

REFERENCES

Avarikioti, Z., Faber, L., Wattenhofer, D.R., n.d. Distributed Computing Group Computer Engineering and Networks Laboratory ETH Zürich.

Bax, F., n.d. Determinization with Monte Carlo Tree Search for the card game Hearts.

Browne, C.B., Powley, E., Whitehouse, D., Lucas, S.M., Cowling, P.I., Rohlfshagen, P., Tavener, S., Perez, D., Samothrakis, S., Colton, S., 2012. A Survey of Monte Carlo Tree Search Methods. IEEE Trans. Comput. Intell. AI Games 4, 1–43. https://doi.org/10.1109/TCIAIG.2012.2186810

Chaslot, G.M.J.-B., Winands, M.H.M., Van Den Herik, H.J., 2008. Parallel Monte-Carlo Tree Search, in: Van Den Herik, H.J., Xu, X., Ma, Z., Winands, M.H.M. (Eds.), Computers and Games, Lecture Notes in Computer Science. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 60–71. https://doi.org/10.1007/978-3-540-87608-3_6

Cowling, P.I., Powley, E.J., Whitehouse, D., 2012. Information Set Monte Carlo Tree Search. IEEE Trans. Comput. Intell. AI Games 4, 120–143. https://doi.org/10.1109/TCIAIG.2012.2200894

Di Palma, S., Lanzi, P.L., 2018. Traditional Wisdom and Monte Carlo Tree Search Face-to-Face in the Card Game Scopone [WWW Document]. arXiv.org. https://doi.org/10.1109/TG.2018.2834618

How many Tic-Tac-Toe (noughts and crosses) games? [WWW Document], n.d. URL http://www.se16.info/hgb/tictactoe.htm (accessed 1.11.25).

Kelly, R., Churchill, D., n.d. Comparison of Monte Carlo Tree Search Methods in the Imperfect Information Card Game Cribbage.

Kocsis, L., Szepesvári, C., 2006. Bandit Based Monte-Carlo Planning, in: Fürnkranz, J., Scheffer, T., Spiliopoulou, M. (Eds.), Machine Learning: ECML 2006, Lecture Notes in Computer Science. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 282–293. https://doi.org/10.1007/11871842_29

Lanctot, M., Lisý, V., Winands, M.H.M., 2014. Monte Carlo Tree Search in Simultaneous Move Games with Applications to Goofspiel, in: Cazenave, T., Winands, M.H.M., Iida, H. (Eds.), Computer Games, Communications in Computer and Information Science. Springer International Publishing, Cham, pp. 28–43. https://doi.org/10.1007/978-3-319-05428-5_3

MCTS Tic-Tac-Toe Visualization [WWW Document], n.d. URL https://vgarciasc.github.io/mcts-viz/ (accessed 1.11.25).

Moravčík, M., Schmid, M., Burch, N., Lisý, V., Morrill, D., Bard, N., Davis, T., Waugh, K., Johanson, M., Bowling, M., 2017. DeepStack: Expert-level artificial intelligence in heads-up no-limit poker. Science 356, 508–513. https://doi.org/10.1126/science.aam6960

Shannon number, 2025. . Wikipedia.

Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., Chen, Y., Lillicrap, T., Hui, F., Sifre, L., Van Den Driessche, G., Graepel, T., Hassabis, D., 2017. Mastering the game of Go without human knowledge. Nature 550, 354–359. https://doi.org/10.1038/nature24270

Świechowski, M., Godlewski, K., Sawicki, B., Mańdziuk, J., 2023. Monte Carlo Tree Search: A Review of Recent Modifications and Applications. Artif. Intell. Rev. 56, 2497–2562. https://doi.org/10.1007/s10462-022-10228-y

Whitehouse, D., Powley, E., Cowling, P., 2011. Determinization and information set Monte Carlo Tree Search for the card game Dou Di Zhu. Presented at the 2011 IEEE Conference on Computational Intelligence and Games, CIG 2011, pp. 87–94. https://doi.org/10.1109/CIG.2011.6031993

APPENDICES

Appendix A: Source Code

The full source code for the DMCTS implementation and related game environment can be found at the following link:
Odyssey Engine - MCTS Branch

Appendix B: Statistical Data

The raw data collected during simulations, including average scores, win rates, and timing metrics, are stored in an Excel file. This file is available for download here:
Simulation Data Excel Sheet

The graphs are linked to a drop-down list, but it will not appear in the Google sheet because you will not have edit permissions. You can add a copy to your own drive to modify the graphs or download an view in Microsoft Excel (recommended).

Appendix C: Questionnaire Responses

Responses from participants who evaluated the AI's performance are compiled in a Google Spreadsheet, which can be accessed at the following link:
Participant Questionnaire Responses