Challenging Human Supremacy: Evaluating Monte Carlo Tree Search and Deep Learning for the Trick Taking Card Game Jass

Joel Niklaus[1, 4, 5], **Michele Alberti**[1], **Rolf Ingold**[1], **Markus Stolze**[2], **Thomas Koller**[3, 4]
[1]Document Image and Voice Analysis Group (DIVA), Fribourg, Switzerland, {firstname}.{lastname}@unifr.ch
[2]User-Interface Software Technology Lab (UIST-Lab), Rapperswil, Switzerland, markus.stolze@hsr.ch
[3]Algorithmic Business Research Team (ABIZ), Lucerne, Switzerland, thomas.koller@hslu.ch
[4]Equal Contribution [5]Contact Author

Abstract

Despite the recent successful application of Artificial Intelligence (AI) to games, the performance of cooperative agents in imperfect information games is still far from surpassing humans. Cooperating with teammates whose play-styles are not previously known poses additional challenges to current state-of-the-art algorithms. In the Swiss card game Jass, coordination within the two opposing teams is crucial for winning. Since verbal communication is forbidden, the only way to transmit information within the team is through a player's play-style. This makes the game a particularly suitable candidate subject to continue the research on AI in cooperation games with hidden information. In this work, we analyse the effectiveness and shortcomings of several state-of-the-art algorithms (Monte Carlo Tree Search (MCTS) variants and Deep Neural Networks (DNNs)) at playing the Jass game. Our key contributions are two-fold. First, we provide a performance overview for state-of-the-art algorithms, thus, setting a strong foundation for further research on the subject. Second, we implement and open-source[1] a platform where different agents (both humans and AI) can play Jass in an automated fashion, effectively reducing the overhead for other researchers who want to perform further experiments.

1 Introduction

In recent years, numerous breakthroughs have taken place in the field of research for AI in games. In particular, in the perfect information games division — where all players are familiar with the entire game state at all times — computers prevail over skilled human players on various occasions, such as Chess (Campbell, Hoane, and hsiung Hsu 2002), the Atari games (Mnih et al. 2015) or Go (Silver et al. 2016).

When it comes to imperfect information games — where players do not know some of the information, such as in card games — there is a thin line separating AI from people who still have the upper hand over state-of-the-art bots. Hidden information is also present in many real-world scenarios, such as business, negotiations, physics, surgery, and others. Many of these situations can be formalized as games that, in turn, can be solved using the methods developed in the card games testbed.

[1]https://gitlab.enterpriselab.ch/jass/info/

Recent work has shown that the distance between humans and AI is becoming smaller in constrained situations. This is particularly evident when considering developments on Texas hold'em no-limit poker (Brown and Sandholm 2018), (Brown and Sandholm 2019) and the StarCraft II computer game (Vinyals et al. 2019). The multiplayer computer game Dota 2 includes hidden information and team play. Although OpenAI Five won against world champions in a 5 vs. 5 game, collaboration remains as an open research area in AI[2].

To instill AI systems with collaboration, card games may be a very suitable testbed since they a) include hidden information, b) frequently have a collaborative aspect, and c) are computationally easy to simulate because they have a finite set of actions.

Motivation

Jass is a trick-taking card game featuring hidden information. In the 4-player Schieber variant, good coordination within the team is crucial for achieving victories in top tournaments.

DeepMind introduced the Hanabi Challenge, opening a new frontier in AI research using the fully cooperative card game Hanabi (Bard et al. 2019). In Hanabi, the players need to lay down cards in order having only the knowledge of the other players' cards. Therefore, the players need to work together to be able to win the game. Jass combines both a cooperative aspect as it includes two competing teams of two cooperating players.

Since there are approximately $1.16e28$ states in Schieber after the cards have been dealt (see Section 2.2) and additionally it is not known in what state the player is because of the hidden cards, the game is computationally complex.

Jass is a very popular Swiss card game and is closely associated with Swiss culture. It is also similar to other games like the American Spades, the British Bridge, and the German Skat. Thus research in Jass is valuable for many other domains.

Contributions

- We perform an analysis of the most promising state-of-the-art methods for AI in card games (Determinized Monte Carlo Tree Search (DMCTS), Information Set

[2]https://openai.com/blog/how-to-train-your-openai-five/

Figure 1: This figure depicts a trick in the Schieber variant. The order of play is counter-clockwise (Harry started with trump diamond 8). Ron had to follow suit with his Diamond 6 and Hermine did not have any Diamonds left, since she played Spades 6. Ginny's card (Diamond Jack) wins this trick because it is the highest trump card. The picture is taken from our Jass server.

Monte Carlo Tree Search (ISMCTS), DNN and Rule-Based (RB)).

- We lay the groundwork for further research on AI in Jass.
- We release public open-source software infrastructure (see Section 5.2) and an Application Programming Interface (API), so anyone can quickly connect their bots and test them both against other bots as well as against human via a GUI.

2 The Jass Game

2.1 Jass in a Nutshell

Jass is a traditional Swiss card game that is trick-taking and often played at social events. It involves hidden information, both a cooperative (cooperation within the team) and a non-cooperative aspect (two opposing teams), is sequential, finite, and constant-sum (since in each game there are always 157 points).

The Schieber variant — our testbed — is one of the most widely played variants of Jass in Switzerland. It is played with two opposing teams of two players each. Each round consists of a trump selection phase and a consecutive card play phase. Since choosing a trump is a significant advantage, tournaments are played by a fixed number of rounds (divisible by 4) so that each player can choose trump an equal number of times. Trump selection implies that the selecting player can determine one of the four suits as trump or alternatively no trump with the card precedence top down or bottom up respectively. The player can also decide to pass on the decision to his teammate. This is called "schieben" and gave the name to the game. An example Schieber trick is shown in Figure 1.

Terminology 4 played cards are called a *trick*, 9 tricks are a *round* (all 36 cards played) and a *game* lasts for 1000 points, or in tournaments for a number of rounds. When a player *passes* in trump selection, the partner can nominate the trump.

The Swiss Intercantonal Lottery provides a guide for general Jass rules[3] and for the variant Schieber specifically[4].

2.2 Complexity

In Schieber, the number of possible paths through the game tree is $36! = 3.72e41$ since there are 36 cards in the game because every card is only played once, and the order matters.

At the beginning of the game the cards are being dealt randomly to the players. There are $\binom{36}{9}\binom{27}{9}\binom{18}{9}\binom{9}{9} = 2.15e19$ possibilities to distribute the 36 cards to 4 stacks. After the cards have been dealt, each player knows their cards, so the possible distributions of the other cards are $2.28e11$.

To estimate the number of possibilities that a round can be played, we gathered empirical evidence from 1.8 million played rounds (see Section 5.1 for the data) to determine the number of valid plays permitted by the rules for each of the 36 cards played. We found $5.1e16$ possible playouts, so the number of states that an algorithm has to deal with after receiving the cards is in the order of $5.1e16 \cdot 2.28e11 = 1.16e28$.

This brings more possibilities and thus makes card play harder.

3 Related Work

In this section, we provide a short overview of the research done in AI development for card games with hidden information and cooperative elements

Sievers and Helmert (2015) applied Upper Confidence Bound for Trees (UCT) to Doppelkopf reaching par-human (on par with average humans) performance. We compare the performance of UCT with a DNN implementation at the example of the similar trick-taking card game Jass.

Applying DMCTS to the multiplayer games Spades and Hearts, Sturtevant (2008) reported similar performance to the state-of-the-art at that time in Spades and slightly better performance in Hearts. He noted that random rollouts outperformed RB rollouts. Similarly, we investigate the effectiveness of random rollouts in comparison to RB rollouts and also the output of the value function of a DNN trained on data obtained from human games.

Whitehouse, Powley, and Cowling (2011) applied DMCTS and ISMCTS to Dou Di Zhu reaching comparable performance. In this work, we compare different configurations of DMCTS with different configurations of ISMCTS and additionally with DNNs and RB baseline algorithms in the card game Jass.

[3] www.swisslos.ch/en/jass/informations/jass-rules/principles-of-jass.html

[4] www.swisslos.ch/en/jass/informations/jass-rules/schieber-jass.html

Using ISMCTS, Watanabe and Lanzi (2018) presented a high-human (on par with the best humans) AI for the Italian card game Scopone consistently beating strong RB players.

Whitehouse et al. (2013) found a MCTS player to be stronger than RB players in the card game Spades. Integrating ISMCTS with knowledge-based methods, they created a more engaging play. Similarly, we compare a DMCTS player with a RB player.

Niklaus et al. (2019) provided an overview of current state-of-the-art methods and discussed them from a theoretical point of view. In contrast, in our work, we implement the most prominent methods (DMCTS, ISMCTS, DNN, and RB) and show their different strengths and shortcomings.

Super-human performance has not yet been achieved in the current state-of-the-art in AI for card games with hidden information and cooperative aspects. In none of these games, a complete analysis of the relevant methods has been carried out. To the best of our knowledge, there has not yet been presented any general AI capable of achieving high performance in more than two of these games.

4 Methods

4.1 Monte Carlo Tree Search

In their literature review, Niklaus et al. (2019) found MCTS variants to be the most promising methods for trick-taking card games like Jass. MCTS is a successful algorithm for perfect information games(Browne et al. 2012). It incrementally builds a search tree for the next few actions and estimates the value of a new node by simulating the game to the end using a rollout policy. Over time, the value of a node becomes the average of all the simulations that passed through it. The decision which part of the tree to extend uses a tree policy that balances exploration and exploitation, guided by the exploration hyper parameter c.

Browne et al. (2012) provide a detailed MCTS family overview. MCTS cannot directly be applied to imperfect information games. DMCTS (Section 4.2) and ISMCTS (Section 4.3) are two major extensions addressing this problem.

4.2 Determinized Monte Carlo Tree Search

A common approach for imperfect information games is to assign values to the unobservable variables and then apply perfect information search methods. For MCTS this sampling is called *determinization*, leading to DMCTS. By sampling and searching multiple states, a more accurate evaluation can be performed. For each sampling, MCTS is performed using a number of iterations.

DMCTS shows the most promising results for imperfect information trick-taking card games according to the literature (Niklaus et al. 2019). It can be parallelized in different ways (leaf, root or tree (Chaslot, Winands, and van den Herik 2008)) and thus scales well on multi-core architectures (Browne et al. 2012). We use root parallelization, where multiple trees are in memory at the same time.

4.3 Information Set Monte Carlo Tree Search

Another possibility to adapt MCTS to imperfect information games is ISMCTS(Cowling, Powley, and Whitehouse 2012). In ISMCTS only a single tree is used for all determinizations and each node in the tree represents an *information set*. The information set captures all states of the games that are indistinguishable for the current player based on his knowledge of the game. This results in a tree where the children are determined not by a single determinization, but by all possible determinizations encountered so far.

To incorporate ISMCTS in the MCTS algorithms, first a determinization is calculated like in DMCTS. Then in the selection phase of MCTS, only children valid with the current determinization are considered. If a node with unvisited children corresponding to the determinization is encountered, that node is expanded. For each determinization, a number of MCTS iterations are performed.

An advantage of ISMCTS is that only a single tree is used so that in later determinization, we expect parts of the tree to be reused, whereas DMCTS starts with a new tree each time. On the other hand, the branching factor of the tree in ISMCTS gets much larger, making it harder to obtain a deeper search tree.

4.4 Deep Neural Network

In contrast to search based methods, Supervised Learning (SL) methods use data from played games to directly learn the best actions or to determine the expected result from an action. For SL we use a DNN, explained in more detail in Section 6.3.

5 General Setup

5.1 Data Sets

Data for training and evaluation was taken from an online platform[5], where users can play the game and either register or play anonymously.

The collected data is from a period of 6 months, starting in October 2017 and consists of about 1.8M played rounds. Each round consists of playing 36 cards, and only completed rounds were taken. The data is split into training, validation, and test sets with a ratio of 0.6:0.2:0.2 by random selection. As plays from the same round are correlated, we further split the files into records for single card plays and shuffle them randomly. From this data set, we filtered out plays by all players that performed less than average, i.e., did not get an average score of 78.5 points. This also eliminates the unregistered players who performed with 78.43 points on average. The resulting data set contains about 14M card plays in the training set and about 4.8M card plays in the test and validation sets.

5.2 Technical Infrastructure

We publish repositories for a Jass server (deployment[6] and sources[7] that can run games and tournaments and display the results, as well as a Python development kit to implement algorithms.

[5] https://www.swisslos.ch

[6] https://jass-server.abiz.ch

[7] https://gitlab.enterpriselab.ch/jass/info/

Any bot implementing a REST API[8] can be connected to the Jass server. We also provide a GUI[9] allowing humans to play on the Jass server[10].

5.3 Tournament Setup

Friendly Jass matches are played until an agreed number of points is reached. Tournaments, however, are usually played for a number of rounds, and the number of points over all rounds are accumulated.

In many card games like Jass, Bridge and Skat, cards are dealt at random in the beginning, and it is much easier to get more points with a good hand than with a bad one. This randomness makes it hard to compare the absolute strength of players.

We address this issue in our experiments by dealing the cards dealt to the North/South pair in the first game to the East/West pair in the second round, which we call a *double round*. We compare the performance of two bots against each other by playing 10 times 100 rounds (= 50 double rounds) and report the mean and the Standard Deviation (STD) of the accumulated score over the 100 rounds.

For more reliability, we disabled additional points awarded to card combinations like Melds (Weisen) and Marriages (Stöck) as well as the Matchbonus (100 points if a team wins all tricks in a round).

6 Implementation

6.1 Time-based DMCTS

The implementation of the Time-based Determinized Monte Carlo Tree Search (T-DMCTS) is publicly available on Github[11]. It uses a ranked RB trump selection. If no trump surpasses a given threshold, it passes.

It uses a time budget as a termination criterion for the search, so it can easily be compared to other bots with the same resources. We use *robust child* as final selection policy, choosing the most visited node after the search. It does not use any heuristic function in the tree policy, does not prune branches, does not bound the scores and uses the standard exploration constant $\sqrt{2}$, since different configurations did not improve the performance in our hyper parameter search.

It uses a determinization factor of 15 and runs for 10s per move. This factor multiplied with the number of tricks remaining in the round results in the number of determinizations created. Example: In the third trick, six tricks are remaining (the current trick included). This means that in this trick, it generates $6 \cdot 15 = 90$ determinizations. For each one we create a thread which after the search returns a selected move together with a score for this move. Then we bundle together all of the determinizations for the same moves and average the scores. Multiplying the average score of a move with the number of times it has been selected gives us a final score. This final score is then used for the final selection. This comes with the advantage that not only the number of times a specific move is selected is considered but also the estimated score it is associated with.

6.2 Iteration-based DMCTS and ISMCTS

The Iteration-based Determinized Monte Carlo Tree Search (I-DMCTS) uses a configurable number of determinization and MCTS iterations, independent of the time budget, to enable testing of different configurations. Results for different numbers of determinization and iterations are given in Section 8.1. The ISMCTS implementation uses the same framework as the I-DMCTS.

6.3 Deep Neural Network

Card Play Network We trained a DNN to perform 3 different tasks using SL. We a) predict the action a, i.e. the card played by a player as a policy function $p(a|s_o)$, b) the value function $v(s_o)$ and c) the card distribution probability $p_{\text{card}}(\text{player} = i|c)$ that the card c was in the hand of player i at the beginning of the game. s_o describes the current state of the game observable by the player.

A single convolutional neural network was used with 3 different heads and loss functions. The loss function for the policy head is the cross entropy loss between the predicted action probability p and the actual action a. The loss of the value prediction is the mean squared error between the predicted value v and the actual result z of the game and finally the loss of the card prediction is the sum of the cross entropy losses between the predicted and actual card distributions for each player.

The DNN consists of 6 convolutional layers with 256 channels and kernel sizes of $2x3$ for the first 3 layers, and $1x2$, $1x2$ and $1x2$ for the following layers, whereby each layer reduces the size of the input as valid padding is used. The 6 layers result in a $1x256$ vector that connects to each loss function by a fully connected layer. The total loss is calculated as the weighted sum of the 3 losses, whereas the weights are chosen so that they scale the magnitude of each loss in the same range i.e., each loss will contributes the same.

The input to the DNN is a $4x9$ matrix of 43 channels containing all the information available to the player. This consists of the cards that have been played so far and in which trick and by which player, the cards in the hand of the current player and the valid cards to play, as well as global information about the game, i.e. who declared trump, how many cards have been played so far and how many points has each team achieved.

Training was done for 200 epochs using an Adam optimizer. R2 regularization was enabled on all weights, but no dropout or batch normalization was used. Training achieved an accuracy of 0.776 for the policy, and 0.771 for the card distribution and Mean Squared Error (MSE) of 0.016 for the value function.

Convolutional networks outperformed similar fully connected networks using the same input by 4% in card accuracy, 2% in the card distribution and 10% in the MSE for the value function.

We use the card-distribution prediction in variants of the search algorithms as P-DMCTS and P-ISMCTS to draw cards

[8] https://jasschamp.ch/wp-content/uploads/2019/09/JassInterface.pdf

[9] https://github.com/JoelNiklaus/jass-server

[10] https://jassteppich.abiz.ch

[11] https://github.com/JoelNiklaus/JassTheRipper

according to the predicted distribution during determinization. The value function is used as a card play algorithm by evaluating all valid cards and selecting the one with the highest value.

Trump Selection Network Trump selection was trained by a different DNN that uses only the cards in the hand of a player as input, as well if the player is the first or second player of the team to be asked to declare trump. The network consists of two fully connected layers with 592 channels followed by a fully connected layer with 7 channels, which corresponds to the 7 possible actions (4 colors, top-down, bottom-up and passing). The accuracy of the network reached 0.8193 on the validation set. Deeper networks did not perform better.

7 Value Estimation Comparison

In this section we compare different methods to estimate the value of a current game state. We assume that estimating the value better leads to a better overall card play performance.

After DMCTS samples a determinization, the algorithm finds itself in a perfect information game situation, so that all the cards are known. To evaluate algorithms in this setting, we omit the sampling and give them the perfect information of the card distribution. The experiment is performed on the on the validation set (described in Section 5.1).

As a baseline comparison, we plot the value estimation from the DNN, which, however, only uses information about the cards in the hand of the current player. The DNN Max. Policy is used to compute a heavy rollout, resulting in a score of the game at the end of the round. The average of a different number of random rollouts (without any tree search!) is also listed. Finally, different numbers of MCTS iterations with random rollout are shown. To calculate the estimated value, we multiply the probability of an action with the value of that action and sum this for all actions. All the methods have access to the hidden information except DNN Value.

Figure 2 shows the results of the different investigated methods. While the improvement from 25 random rollouts to 10 is evident, the improvement of 1000 to 25 random rollouts is only marginal. The DNN value function seems to be comparable to the average of 10 random rollouts. The policy function does not give significantly better results than using random rollout with 1000 iterations or 100 MCTS iterations. The accuracy of the MCTS based value estimation improves clearly with the number of iterations.

Random rollouts do not seem to improve much after a number of iterations have been reached, while MCTS continually improves the accuracy with more iterations. We expect that this improved accuracy translates to stronger overall card play. Overall, already 100 MCTS iterations outperform both 1000 random rollouts and the DNN Max Policy rollout in the perfect information game setting. The difference between the investigated methods is particularly evident in the first few tricks (0 to 24 cards played). Our analyses show that this phase is also the most crucial time in a round. The further the round progressed, the easier it becomes to play optimally and thus, the smaller the difference between different bots.

Table 1: Percentage of total points of I-DMCTS playing against DNN with different number of determinizations and iterations and exploration constant 1.5.

Iter.	Determinizations			
	25	100	1K	10K
25	48.07 ± 0.85	48.76 ± 1.58	49.45 ± 1.00	50.02 ± 1.34
100	49.53 ± 1.10	49.92 ± 0.66	49.89 ± 1.21	49.55 ± 0.97
1K	50.11 ± 1.27	50.30 ± 0.93	50.65 ± 0.66	50.75 ± 1.34
10K	49.98 ± 0.71	50.79 ± 0.89	50.38 ± 1.02	very costly

Table 2: Percentage of total points of different trump selection algorithms playing against DNN.

Bot	Result (%)
Random	34.19 ± 2.02
Simple Rule	47.69 ± 0.82
Ranked Based Rule	49.26 ± 1.11
MCTS	48.23 ± 1.98

8 Experiments Between Bots

In Section 8.1 we describe experiments with DMCTS hyper parameters. Since Jass consists of two distinct phases (trump selection and card play) we can also separately evaluate our bots in these two phases, explained in detail in Section 8.2 and 8.3 respectively. In all of these experiments between bots we used double rounds (see Section 5.3) to reduce randomness.

Note that already a small difference in points between two teams can lead to a victory in a Jass game, like in a ski race a difference of 0.5s can bring an athlete from outside the top 20 to winning the gold medal.

Each experiment consists of playing a game of 100 rounds (= 50 double rounds) 10 times. We report the mean percentage of total points and the STD of the 10 different games. The p-value has been calculated with an unpaired t-test.

8.1 Hyper Parameters for DMCTS

We conducted several experiments to find the best hyper parameters for DMCTS. Given a specified number of iterations to be performed, or a specific time constraint, we investigate if is it better to have a larger number of determinizations, thus exploring many different card configurations, or if it is better to devote more resources to the MCTS giving a more accurate result for the cards.

Determinizations and Iterations Figure 4 shows an overview of the performance of DMCTS with different allocations of a fixed budget against DNN Max Policy. The budget is 800K iterations (e.g., 20K determinizations with 40 iterations each, or 500 determinizations with 1600 iterations each). We find that an increase in the number of determinizations is beneficial up to a certain point ($p = 0.015$ for 1Kx800 and 500x1600). However, further increasing the number of determinizations to over 1K shows no improvement. We find the sweet spot with this particular budget to be at 1K determinizations with 800 iterations each.

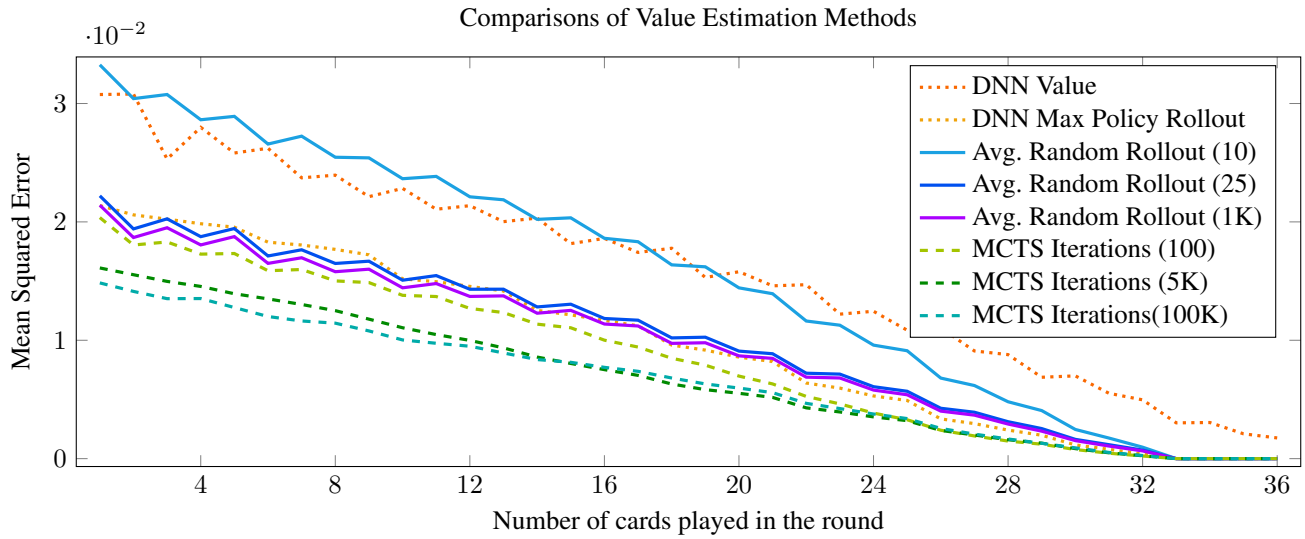
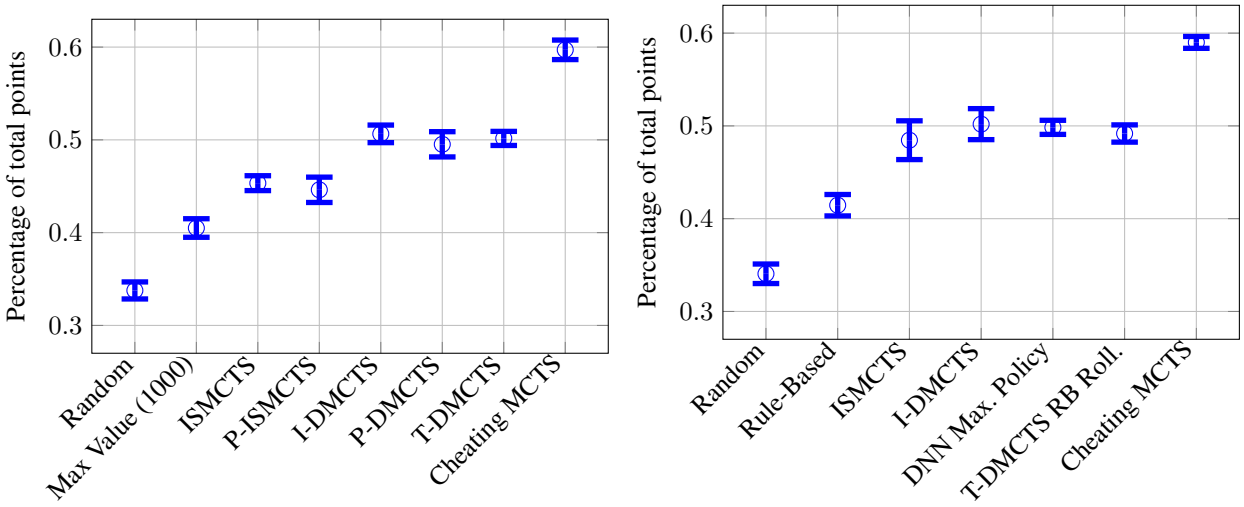


Figure 2: We calculated the MSE between the estimated value from the algorithm and the actual outcome at the end of the round. Since the validation set contains 4.8M card plays in total it contains 133K card plays per game stage (number of cards played) on average. Each data point therefore represents the mean of the MSEs of these 133K card plays.



(a) Against DNN Max. Policy

(b) Against T-DMCTS

Figure 3: Each experiment consists of playing a game of 100 rounds (= 50 double rounds) 10 times, and the average received percentage of total points is shown. The error bar is the STD of the 10 different games.

Exploration Constant In pure MCTS experiments we have found that an exploration factor of 0.2 gives the best results. However, in DMCTS, much larger exploration values than for pure MCTS result in better performance. We find the standard value of around $\sqrt{2}$ to show the highest mean value ($p = 0.13$ for 0.5 and 1.5) This corresponds with the findings of Browne et al. (2012), stating that for perfect information games, very low exploration constants are optimal, but for imperfect information games, the value lies higher.

Scalability In Section 7 we saw that a bigger number of MCTS iterations can increase the accuracy of estimating the value at the end of the game. In this paragraph we present an experiment that checks if more iterations and more determinizations really are beneficial to the overall card play strength (measured in percentage of total points).

Table 1 shows different combinations of iterations and determinizations of DMCTS against DNN which allows us to interpret the scalability properties of DMCTS.

With 25 determinizations there is a strong improvement from 25 to 100 iterations ($p < 0.01$). However, our data does

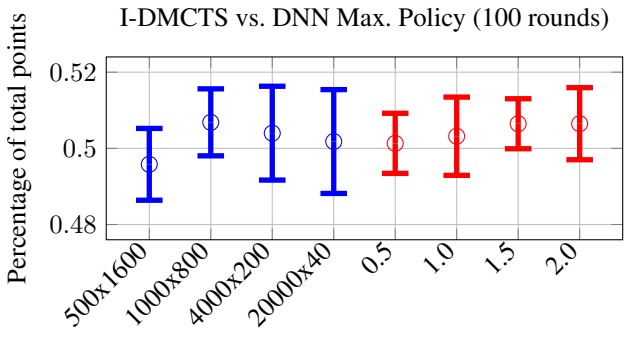


Figure 4: Different configurations of I-DMCTS playing against DNN Max. Policy. The first four results (blue) show different ratios between the number of determinizations d and MCTS iterations i ($d \times i$), while the last 4 results (red) show different exploration parameters c each executed with $d = 1000$ and $i = 1000$.

not clearly support an improvement from 100 to 10K iterations ($p = 0.29$). Yet, for 100 determinizations, there is an increase from 100 to 10K iterations ($p = 0.023$). Increasing both the determinizations and the iterations clearly has a positive effect on the overall card play strength ($p < 0.0001$ for 25x25 to 1Kx1K). When the number of determinizations or iterations are high though (1Kx1K to 10Kx1K and to 1Kx10K), our data does not support clear claims. It would be very interesting to see how the card play strength changes from 1Kx1K to 10Kx10K and further to 1Mx1M. However, running experiments in these dimensions are very costly (10Kx10K would take 600h (25 days) on a 8 core machine running 16 threads).

8.2 Trump Selection Phase

To evaluate the trump selection methods, we let four different trump selection methods play against each other while using the same card play algorithm, DNN, for all of them. DNN card play is fast, robust, and deterministic, putting no additional variance into the experiment.

The four trump selection methods we tested were the following: First, the *Random* chooses the trump completely randomly. The *Simple RB* method tries to estimate the number of certain tricks that can be won. The *Ranked RB* implements a ranking algorithm and is used in T-DMCTS. The *MCTS* method considers the trump selection as just another move in the tree to be searched. Finally, the *DNN* performs trump selection as described in Section 6.3.

The results are shown in the Table 2 for playing 100 rounds 10 times as described in Section 5.3. DNN achieves the best results, while the more elaborate RB algorithm based on the ranking is only slightly worse ($p = 0.063$ for DNN and Ranked RB).

Trump selection proves quite essential, as even a simple algorithm is much better than random selection, so a good bot must combine good trump selection and card play. In Schieber, passing can be very valuable in trump selection, since more information is available afterward. The player who selects trump after the first one passed knows for example that the first player does not have very good cards to choose a trump. So, with passing, the players have another shot at a good trump. We analyzed the choices of the MCTS based trump selection method and noticed that it rarely passes. This may be a reason for it to perform worse than the DNN method ($p = 0.012$).

8.3 Card Play Phase

To evaluate the different card play algorithms, we let them play against each other with the settings described in Section 5.3. The DMCTS and ISMCTS are the bots as described in Section 4. The RB bot[12] is a baseline bot and builds on the Jass Challenge environment released by the Software Engineering company Zühlke[13]. It won the Zühlke Jass Challenge Competition in 2017. The T-DMCTS RB rollouts uses RB rollouts instead of random rollouts. The random bot selects a random card while using DNN for trump selection, the Max Value bot evaluates the value network for each valid card out of 1000 card distributions and plays the card with the highest value. The P-DMCTS and P-ISMCTS bots use the probability distributions of the cards from the DNN and draw cards according to this distribution instead of random cards. The cheating MCTS has access to the hidden information (the cards of the other players) and is added as an upper bound.

Figure 3(a) displays the results of the different bots against the DNN method, while Figure 3(b) compares the strength of different bots against the T-DMCTS method. The bots are configured with their best settings; comparisons between different settings are explored more in the following sections.

As expected, knowledge of the unknown cards is precious, which can be seen in the big jump in strength by the cheating MCTS player. However, surprisingly, having access to the probability distributions of the cards does not improve the card play strength compared to just sampling random cards ($p = 0.046$ for P-DMCTS and I-DMCTS and $p = 0.17$ for P-ISMCTS and ISMCTS). Rather, the variance increases, suggesting that there are both occasions where the DNN guessed the distribution of the cards correctly and others where it did not.

9 Conclusion and Outlook

We provide a comparison of the most widely used methods in trick-taking card games at the example of the Schieber variant of the Swiss card game Jass. In the trump selection phase, empirical evaluation suggests that the DNN slightly outperforms the ranked RB method. In the card play phase, we found that the similarly strong DMCTS and DNN outperform the random baseline, a robust RB bot and also ISMCTS.

Acknowledgments

We thank Swisslos for providing us with the data set of online games.

[12] https://github.com/Murthy10/pyschieber/tree/master/pyschieber/player/challenge_player

[13] https://github.com/webplatformz/challenge

References

Bard, N.; Foerster, J. N.; Chandar, S.; Burch, N.; Lanctot, M.; Song, H. F.; Parisotto, E.; Dumoulin, V.; Moitra, S.; Hughes, E.; Dunning, I.; Mourad, S.; Larochelle, H.; Bellemare, M. G.; and Bowling, M. 2019. The hanabi challenge: A new frontier for AI research. *CoRR* abs/1902.00506.

Brown, N., and Sandholm, T. 2018. Superhuman ai for heads-up no-limit poker: Libratus beats top professionals. *Science* 359(6374):418–424.

Brown, N., and Sandholm, T. 2019. Superhuman ai for multiplayer poker. *Science* 365(6456):885–890.

Browne, C. B.; Powley, E.; Whitehouse, D.; Lucas, S. M.; Cowling, P. I.; Rohlfshagen, P.; Tavener, S.; Perez, D.; Samothrakis, S.; and Colton, S. 2012. A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in Games* 4(1):1–43.

Campbell, M.; Hoane, A.; and hsiung Hsu, F. 2002. Deep blue. *Artificial Intelligence* 134(1):57 – 83.

Chaslot, G. M. J. B.; Winands, M. H. M.; and van den Herik, H. J. 2008. Parallel monte-carlo tree search. In van den Herik, H. J.; Xu, X.; Ma, Z.; and Winands, M. H. M., eds., *Computers and Games*, 60–71. Berlin, Heidelberg: Springer Berlin Heidelberg.

Cowling, P. I.; Powley, E. J.; and Whitehouse, D. 2012. Information set monte carlo tree search. *IEEE Transactions on Computational Intelligence and AI in Games* 4(2):120–143.

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A. A.; Veness, J.; Bellemare, M. G.; Graves, A.; Riedmiller, M.; Fidjeland, A. K.; Ostrovski, G.; Petersen, S.; Beattie, C.; Sadik, A.; Antonoglou, I.; King, H.; Kumaran, D.; Wierstra, D.; Legg, S.; and Hassabis, D. 2015. Human-level control through deep reinforcement learning. *Nature* 518(7540):529–533.

Niklaus, J.; Alberti, M.; Pondenkandath, V.; Ingold, R.; and Liwicki, M. 2019. Overview of artificial intelligence for card games and its application to the swiss game jass. In *2019 6th Swiss Conference on Data Science (SDS)*, 25–30. Bern, Switzerland: IEEE.

Sievers, S., and Helmert, M. 2015. A doppelkopf player based on uct. In *KI 2015: Advances in Artificial Intelligence*, 151–165. Springer International Publishing.

Silver, D.; Huang, A.; Maddison, C. J.; Guez, A.; Sifre, L.; van den Driessche, G.; Schrittwieser, J.; Antonoglou, I.; Panneershelvam, V.; Lanctot, M.; Dieleman, S.; Grewe, D.; Nham, J.; Kalchbrenner, N.; Sutskever, I.; Lillicrap, T.; Leach, M.; Kavukcuoglu, K.; Graepel, T.; and Hassabis, D. 2016. Mastering the game of Go with deep neural networks and tree search. *Nature* 529(7587):484–489.

Sturtevant, N. R. 2008. An analysis of uct in multi-player games. In *Computers and Games*, 37–49. Berlin, Heidelberg: Springer Berlin Heidelberg.

Vinyals, O.; Babuschkin, I.; Chung, J.; Mathieu, M.; Jaderberg, M.; Czarnecki, W.; Dudzik, A.; Huang, A.; Georgiev, P.; Powell, R.; Ewalds, T.; Horgan, D.; Kroiss, M.; Danihelka, I.; Agapiou, J.; Oh, J.; Dalibard, V.; Choi, D.; Sifre, L.; Sulsky, Y.; Vezhnevets, S.; Molloy, J.; Cai, T.; Budden, D.; Paine, T.; Gulcehre, C.; Wang, Z.; Pfaff, T.; Pohlen, T.; Yogatama, D.; Cohen, J.; McKinney, K.; Smith, O.; Schaul, T.; Lillicrap, T.; Apps, C.; Kavukcuoglu, K.; Hassabis, D.; and Silver, D. 2019. Alphastar: Mastering the real-time strategy game starcraft II. [Online; accessed January 29, 2019].

Watanabe, M. N., and Lanzi, P. L. 2018. Traditional wisdom and monte carlo tree search face-to-face in the card game scopone. *IEEE Transactions on Games* 10:317–332.

Whitehouse, D.; Cowling, P. I.; Powley, E. J.; and Rollason, J. 2013. Integrating monte carlo tree search with knowledge-based methods to create engaging play in a commercial mobile game. In *Proceedings of the Ninth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, AIIDE'13, 100–106. AAAI Press.

Whitehouse, D.; Powley, E. J.; and Cowling, P. I. 2011. Determinization and information set monte carlo tree search for the card game dou di zhu. In *2011 IEEE Conference on Computational Intelligence and Games (CIG'11)*, 87–94. IEEE.