Improving Determinized Search with Supervised Learning in Trick-Taking Card Games

by

Christopher Solinas

A thesis submitted in partial fulfillment of the requirements for the degree of

Master of Science

Department of Computing Science

University of Alberta

Abstract

Current state-of-the-art algorithms for trick-taking card games use a process called determinization. Determinization is a technique that allows the application of perfect information state evaluation algorithms to imperfect information games. It involves a two-step process in which a perfect information variant of the game state is sampled from the player's information set and then solved using an algorithm like minimax search.

The majority of recent work related to determinization has focused on addressing some of the theoretical flaws tied to using perfect information techniques to play imperfect information games. However, these works have largely neglected another important part of the equation: inference. Inference involves estimating the state probability distribution of an information set using state information like past opponent actions. It lets players of trick-taking card games predict which cards opponents are holding based on the cards that have been played so far. Inference is crucial for the performance of algorithms that use determinization because it allows states to be sampled according to a better estimate of the true state probability distribution in the information set. This results in improved estimates for action values.

In this thesis, I investigate inference in trick-taking card games. In particular, I present a technique that uses past actions to predict hidden state information like the locations of individual cards. I show that deep learning can be useful for handling the larger input feature spaces associated with a richer

state representation, and lastly, I explain how to combine these predictions to estimate the probability distribution of states within an information set and improve determinized search techniques — leading to a new state-of-the-art in computer Skat.

Preface

Portions of Chapter 4 and 5 were accepted for publication in January 2019 as a main technical track conference paper titled: C. Solinas, D. Rebstock, and M. Buro, "Improving Search with Supervised Learning in Trick-Based Card Games," Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence. I was responsible for designing and implementing the main contribution and gathering empirical results. Douglas Rebstock provided ideas and advice throughout and implemented and collected empirical results for a vital performance metric. This work was done under the supervision of Professor Michael Buro.

Acknowledgements

I would like to thank my supervisor, Professor Michael Buro, for his advice and help with research, writing, and teaching. I appreciate the time and dedication that his hands-on approach with students requires. It has undoubtedly been helpful in all aspects of life as a graduate student.

I would like to thank Douglas Rebstock and Professor Nathan Sturtevant for their helpful ideas over the course of our collaboration. I also appreciate advice from Marius Stanescu, Arta Seify, Professor Levi Lellis, and Shuyi Zhang.

Finally, I would like to acknowledge that this research was enabled in part by Compute Canada and financial support from the Government of Alberta.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The long term goal of many artificial intelligence (AI) researchers is to develop agents that can seamlessly interact with and solve general problems in the real world. The real world is complex, so this is a lofty ambition that modern algorithms seem distant from achieving. Nevertheless, historical progress has been remarkable. AI researchers have advanced the field by developing new techniques that enable intelligent decision-making in successively more difficult tasks. Many have used games as an application domain because games provide a controlled environment for studying decision-making.

As computer agents successfully play more complicated games, their underlying algorithms can be applied to more complicated practical problems. For instance, Monte Carlo Tree Search [Bro+12] (MCTS) was originally developed to play *Go*, but has since been applied by the European Space Agency to develop a state-of-the-art algorithm for planning the interplanetary trajectories of spacecraft [HI15]. In another recent example, OpenAI applied Proximal Policy Optimization [Sch+17] (PPO) to the popular multiplayer game *DOTA 2*. Later, they found that the same algorithm could be used to control robotic hands with unprecedented dexterity [Ope18].

Historically, prominent examples of success for AI in games has mostly been achieved in the two-player perfect information domain — where all game state information is visible to all players. Algorithms are able to perform forward simulation to evaluate potential moves while considering every response available to the opponent; in small games they can simulate all the way to the

end of the game while calculating all possible outcomes for each of their available actions. The main challenge in these domains is developing algorithms that can play larger games with many possible actions per decision point and many decision points before the outcome is decided. Landmark achievements have led to algorithms able to play successively larger games at a high level. These include the solving of Checkers [Sch+07] and superhuman play in Chess [CHH02], Backgammon [Tes95], Othello [Bur97], and Go [Sil+16; Sil+17b]. In a recent monumental success, the AlphaGo Zero algorithm [Sil+17b] surpassed the human level of play in Go, Chess, and Shogi through self-play without any human knowledge. Go is considered the most difficult two-player perfect information game for AI, and AlphaGo Zero's performance has led many to proclaim AI supremacy in perfect information games.

Imperfect information games are more complex than perfect information games because players must account for the probability distribution over opponents' private information. For instance, in most card games, which are very popular among human players, players must consider the private cards of their opponents to make good decisions. As such, different algorithms are necessary for good performance in these games.

Many widely-used techniques for imperfect information games, like Counterfactual Regret Minimization [Zin+08] in poker, look to solve the game in its entirety before real play. This works well for smaller variants of poker, but the computation becomes intractable in larger ones. In larger games, the standard approach is to construct an abstraction that reduces the size of the game and reason about decision-making on the smaller version. This can lead to situations where performance in the original game is sacrificed to reduce the size and complexity of the abstracted version of the game [Zin+08].

In trick-taking card games like Contract Bridge or Skat, good abstractions are difficult to construct. Grouping certain cards together, for instance, is challenging because the outcome of an action often depends on the precise holdings of each player. Current state-of-the-art algorithms instead implement a two-step process called *determinization*, which samples a perfect information

2

version of the state and then solves for, or estimates, the best action as if playing a perfect information game. This approach has proven effective, but there are limitations [Lon+10]. First, the best action in the perfect information variant of the game is not guaranteed to be the best in the real game. Most recent prior work in trick-taking card games has focused on this problem, but there is another important issue that must be accounted for. That is, the perfect information states that are consistent with the player's knowledge of the state are not all equally likely, given what has happened in the game so far. Opponent actions can reveal additional information about the cards in their hand; players perform *inference* by adjusting their beliefs about opponent cards based on this information. Good human players perform inference to gain an edge on their opponents, but current state-of-the-art computer players only partially consider it.

In this thesis, I investigate inference in trick-taking card games. I show that past actions can be used to predict fine-grained information like the locations of individual cards. Previous table-based methods for inference use limited representations of the full state. Incorporating action history into the state representation can be beneficial for card location prediction but leads to a large input feature space. However, I show how to learn an abstraction of this large feature space that leads to better card location predictions using deep neural networks. Finally, I show that combining card location predictions leads to an estimate of the likelihood of each determinized state. These estimates can be used to improve the playing strength of current state-of-the-art determinized search techniques in the domain of Skat.

The rest of this thesis is organized as follows. First, the background required to understand this work is provided, including notations, definitions, and the rules of Skat. Next, previous work on imperfect information games, and specifically trick-taking card games, is explained. From there, I describe a novel technique for predicting the locations of individual cards in Skat that leverages action history from the cardplay phase. This is followed by a section explaining how to use card location probabilities to perform state inference

and make better decisions in existing algorithms for trick-taking card games. In both sections, I provide experimental results that show the effectiveness of these contributions in Skat. Next, I propose and test extensions to individual card location prediction. I finish with conclusions and ideas for future research.

Chapter 2

Background

This chapter provides the background material necessary to understand the contents of the rest of this thesis. It combines terminology from game theory and heuristic search. It also introduces trick-taking card games and explains Skat — the principal application domain of this work.

2.1 Imperfect Information Games

Perfect information games like Tic-Tac-Toe, Chess, or Go are games in which the precise state is visible to all players. This means that a player can choose actions specific to each possible node in the game tree.

As an example, Figure 2.1 shows the game tree for *The Sharing Game.* The game involves 2 players deciding how they will split 2 dollars. Both players want to end the game with the largest share possible. Player 1 gets to choose the first action and must offer Player 2 a deal. The deal can be one of three
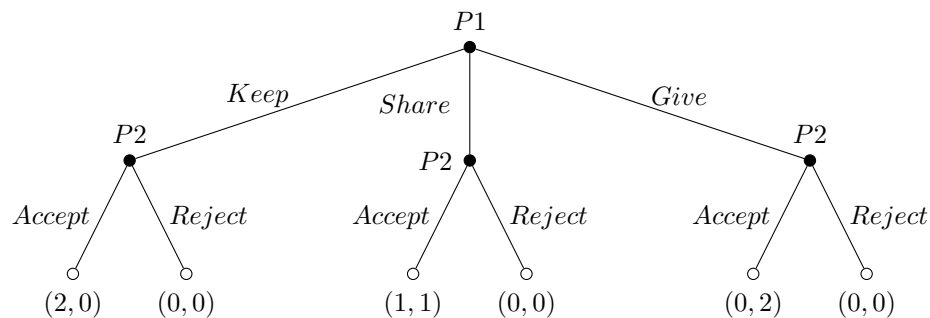
Figure 2.1: Game tree for *The Sharing Game.* Player 2 knows the exact state of the game and can therefore specify an action at every node.

options: Player 1 gets to *Keep* both dollars, the players *Share* the dollars evenly, or Player 1 can *Give* both dollars to Player 2. Player 2 can then look at the deal and decide whether or not to *Accept* or *Reject* it; Player 2 is free to choose a different action depending on which deal they are offered.

Decision making in imperfect information games is fundamentally different than in perfect information games due to the presence of information sets: sets of states that are indistinguishable to the player to move. Players must decide how to act in information sets rather than cherry-picking actions in individual states. Simple simultaneous move games like *Rock-Paper-Scissors* are considered imperfect information because players are unable to "see" the move of their opponent before taking an action. Figure 2.2 shows the game tree for *Rock-Paper-Scissors*: a simple game with a single decision point for each player, where both players can choose to play either R, P, or S. Player 1 starts by choosing an action, but that action is not observed by Player 2. This means that Player 2's strategy cannot be different for any of the 3 possible states that form the information set in the game tree. In other words, Player 2 cannot choose R every time Player 1 chooses S and S every time Player 1 chooses P. However, Player 2 must consider the result of taking an action in any of the possible states in the information set to build a good strategy.
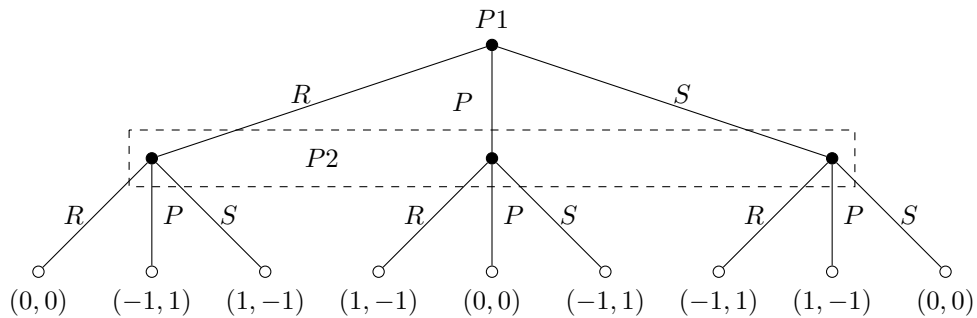
Figure 2.2: Game tree for *Rock-Paper-Scissors*. Player 2 cannot distinguish between states inside the information set (dotted rectangle) and must play the same strategy for all 3 states.

2.2   Trick-Taking Card Games

Trick-taking card games are a special class of imperfect information games. They are played in various formats around the world, and many variants have annual international tournaments for professional players. Some of the many examples are Skat, Contract Bridge, Hearts, and Spades.

After a random deal, play typically starts with a bidding phase in which players compete to see which of them will declare the scoring and winning conditions for that game. Cardplay follows bidding and consists of several tricks in which players each play a card in clockwise order. After each player has played a card, the trick is completed and belongs to the player who played the highest ranked card according to game-specific rules. The trick winner typically "leads" the next trick by playing the first card. In many trick-taking card games, players are forced to "follow suit" by playing a card in the same suit as the trick leader if they have one — otherwise any card can be played.

Several properties of trick-taking card games are interesting to AI researchers. Many are played in teams, and successful agents are required to cooperate with another player. Players reveal hidden information with every move. This provides opportunities for all players; they can try to infer the remaining cards of the player who made the move. Players can be forced to play specific cards in certain situations, and other players must identify when this is the case. Players can also choose to play moves that will force other players to reveal information about their hands with their follow-up actions. Furthermore, a player can deceive opponents by taking actions that cause the opponent to hold incorrect beliefs about the card distribution. A bit like bluffing in poker, this can change which actions the opponent takes in later tricks — possibly allowing the player to win from an otherwise losing position.

2.2.1   Skat

Though particularly popular in Germany, Skat is a 3-player trick-taking card game that is played competitively in clubs and tournaments worldwide. Each

player is dealt 10 cards from a 32-card deck, and the remaining two (called the skat) are dealt face down. Players earn points by winning games, which can be broken down into two main phases: bidding and cardplay.

In the bidding phase, players make successively higher bids to see who will become the soloist for the game. Playing as the soloist means playing against the other two players during cardplay and carrying the risk of losing double the amount of possible points gained by a win. The soloist has the advantage of being able to pick up the skat and discard 2 of their 12 cards. The soloist then declares which of the possible game types will be played. These are summarized in Table 2.1. Standard rules include suit games (where the 4 jacks and a suit chosen by the soloist form the trump suit), grands (where only the 4 jacks are trump), and nulls (where there are no trump cards and the soloist must lose every trick to win). In suit and grand games, players get points for winning tricks containing certain cards during the cardplay phase. Aces, tens, kings, queens, and jacks are worth 11, 10, 4, 3, and 2 card points respectively. Unless otherwise stated, the soloist must get 61 out of the possible 120 card points to win suit or grand games.

The score gained or lost by the soloist depends on the game's base value and a variety of multipliers. Jacks are ranked by suit in the following order: clubs, spades, hearts, diamonds. Base values are multiplied by $k + 1$, where k is determined by the player having (or not having) the top k jacks in their hand. Additional multipliers can be gained by declaring one or more of the game type modifiers listed in Table 2.2. The game value (base $\times$ multiplier) is also the highest possible bid the soloist can have made without automatically losing.

Cardplay consists of 10 tricks in which the trick leader (either the player to the left of the dealer in the first trick or the player who won the previous trick) plays the first card. Play continues clockwise around the table until each player has played. Players may not pass and must play a card from the same suit as the leader if they can — otherwise the player is "void" in that suit and any card can be played. The winner of the trick is the player who played

Table 2.1: Skat game type descriptions.

Type	Value	Trumps	Win Condition
Diamonds	9	Jacks and Diamonds	$\geq$ 61 card points
Hearts	10	Jacks and Hearts	$\geq$ 61 card points
Spades	11	Jacks and Spades	$\geq$ 61 card points
Clubs	12	Jacks and Clubs	$\geq$ 61 card points
Grand	24	Jacks	$\geq$ 61 card points
Null	23	No trump	losing all tricks

Table 2.2: Skat game type modifiers.

Modifier	Description
Schneider	$\geq$90 card points for soloist
Schwarz	soloist wins all tricks
Schneider Announced	soloist loses if card points < 90
Schwarz Announced	soloist loses if opponents win a trick
Hand	soloist does not pick up the skat
Ouvert	soloist plays with hand exposed

the highest card in the led suit or the highest trump card. Play continues until there are no cards remaining, at which point the outcome of the game is decided. Players typically play "lists" of 36 or 48 games before declaring the player who amassed the highest score as the overall winner. Many of the details of this complex game have been omitted because they are not required to help understand this work. For a more in-depth explanation about the rules of Skat I refer interested readers to `https://www.pagat.com/schafk/skat.html`.

One of the main challenges with developing search-based algorithms that can play Skat at a level on par with human experts is the number and size of the information sets in the game. For instance, the information set of a player who is leading the first trick as a defender contains over 42 million possible states. Other challenges include playing cooperatively with a partner on defense to defeat the soloist. This requires players to infer their partner's cards, and

human experts resort to intricate signalling patterns to share information.

2.3 Extensive Form Games

When studying decision-making in games, it helps to have explicit represen-
tations of a game's key aspects. A finite extensive form game is a formal
description of a game commonly used in game theory. The representation
that an extensive form game provides is flexible; both perfect and imperfect
information games can be modelled. An extensive form game G is described
by the following tuple:

$$G = \langle N, A, H, Z, \rho, \sigma, u, \mathcal{I} \rangle \tag{2.1}$$

N is the set of players and H is the set of non-terminal states in the game.
In extensive form games, states are equivalent to histories because a game state
is defined as the history of all actions taken from an initial state. Z is the set
of terminal histories. $H \cap Z = \emptyset$ and $H \cup Z = S$ is the set of all states in the
game. Z corresponds to the set of leaf nodes in a game tree. Given $h, j \in S$,
$h \sqsubseteq j$ is used to denote that h is a prefix history to j. This means that h is
visited on the path to j. Utility function $u_i : Z \to \mathbb{R}$ gives a real-valued payoff
to player i given that the game ends at state $z \in Z$. $\rho : H \to N$ is a function
that defines which player is to move in state $h \in H$.

$A(h)$ is the set of actions available in state h. $\sigma : H \times A \to S$ is the state
transition function that maps a non-terminal state and action to a new state.
For all $h_1, h_2 \in H$ and $a_1, a_2 \in A$ if $\sigma(h_1, a_1) = \sigma(h_2, a_2)$ then $h_1 = h_2$ and
$a_1 = a_2$.

Information Sets $I \in \mathcal{I}$ are partitions of non-terminal states that players
cannot tell apart. They satisfy the following constraints: $\forall h, j \in I$, $\rho(h) = \rho(j)$
and $A(h) = A(j)$. The player $\rho(h)$ knows that the current state of the game is
part of I, but is unable to tell h from j. $\forall I \in \mathcal{I}, |I| = 1$ in a perfect information
game.

2.4 Strategies

Also referred to as their policy, a player i's strategy π_i determines how they choose actions in information sets. $\pi_i(I)$ is a probability distribution over $A(I)$ and $\pi_i(I, a)$ is the probability of taking action a in information set I. A strategy profile $\pi = \langle \pi_0, \pi_1, ..., \pi_{n-1} \rangle$, is a tuple consisting of a strategy for each player, and π_{-i} is the tuple of strategies for the opponents of player i.

Strategy profiles lead to reach probabilities $\eta^{\pi}(h)$ which indicate the likelihood of reaching state h given that players play according to π. Reach probabilities can be conditioned on having already reached a prefix; $\eta^{\pi}(z|h)$ is the probability of reaching z under π given that h was reached.

2.4.1 Values

The standard utility function u in extensive form games assigns real-numbered payoffs to every player in terminal game states. However, it can be useful to quantify "how beneficial" a non-terminal state is to each player given that they are collectively following a strategy profile. Values allow a player to compare strategies based on the utility of possible terminal histories in non-terminal states and ultimately decide which actions should be taken throughout the game.

The expected value, given strategy profile π, for each available action can be calculated at non-terminal state $h \in H$ using:

$$v_i(\pi, h, a) = \sum_{z \in Z, h \cdot a \sqsubseteq z} \eta^{\pi}(z) u_i(z) \tag{2.2}$$

State values can be computed by weighing the sum of action values by the probability of taking each action under the player's current strategy π_i:

$$v_i(\pi, h) = \sum_{a \in A(h)} \pi_i(h, a) v_i(\pi, h, a) \tag{2.3}$$

This definition extends to information set values by considering the reach probability of state $h \in I$ given that the player is in information set I.

$$v_i(\pi, I) = \sum_{h \in I} \eta(h|I) v_i(\pi, h). \tag{2.4}$$

Calculating values requires significant computation in large games and can become intractable. In these situations, there are several techniques for estimating values designed so that AI players can play larger games without having to compute exact values.

2.5 Solving Imperfect Information Games

Solving an imperfect information game amounts to computing a Nash equilibrium. A Nash equilibrium exists for every finite extensive form game [Nas50]; it is a strategy profile in which every player's strategy is a best-response to the rest of the strategy profile. Optimal play consists of playing a specific strategy from the equilibrium profile. Assuming that other players play according to the equilibrium profile, deviations can not result in an increase in utility for any player. Equilibriums are not unique and strategies from one equilibrium profile are not guaranteed to be a best-response to strategies from another in many scenarios.

Despite recent advances, modern algorithms are unable to solve the largest games in practice. As a result, algorithms have been designed to find solutions that approximate optimal play.

2.5.1 Counterfactual Regret Minimization

There is a well-known connection between minimizing the online learning concept of regret in a self-play domain and approximating a Nash equilibrium [Zin+08]. Regret can be informally understood as the amount of increased utility that a player would have received had they played optimal actions; reducing regret allows a player to play better, and if regret becomes small enough for both players in a two-player game, then that strategy profile is an approximate Nash equilibrium. However, regret minimization requires significant time and space, and traditional algorithms do not scale well to large games like Poker.

Counterfactual Regret Minimization [Zin+08] (CFR) is an algorithm for

finding sufficiently good approximate solutions in two-player zero-sum imperfect information games. Counterfactual regret is the standard concept of regret applied to counterfactual values. Counterfactual values (Equation 2.6) are similar to the values defined in Equation 2.3, but they capture that player i played with the intention of reaching state h instead of following π (i.e. $\eta_i^\pi(h) = 1$). The key idea is that counterfactual regret can be minimized independently at each information set by controlling only $\pi_i(I)$. As it turns out, minimizing counterfactual regret at each information set minimizes overall regret as well.

$$v_i^c(\pi, h, a) = \sum_{z \in Z, h \cdot a \sqsubseteq z} \eta_{-i}^\pi(h) \eta^\pi(z|h) u_i(z) \tag{2.5}$$

$$v_i^c(\pi, h) = \sum_{a \in A(h), h \sqsubseteq z} \pi_i(h, a) v_i^c(\pi, h, a) \tag{2.6}$$

Recent algorithms based on CFR have led to superhuman play in Heads Up No-Limit Texas Hold'em. DeepStack [Mor+17] uses continual re-solving which is a depth-limited search where terminal counterfactual values can be estimated at non-terminal nodes using a neural network. Libratus [BS17] applies an abstraction-based approach early in the game, and then switches to an approach similar to continual re-solving, called nested sub-game solving, at later decision points.

2.5.2   CFR in Trick-Taking Card Games

Due to its recent success and strong theoretical guarantees, the CFR family of algorithms is the state-of-the-art for many imperfect information games. CFR has variants designed to tackle different challenges across games, but none have been successfully applied to trick-taking card games. Trick-taking card games are challenging for CFR-based algorithms in both the theoretical and practical sense.

Many of CFR's theoretical guarantees, such as upper bounds on regret, no longer apply to 3-player or non-zero sum games. In general, CFR does not produce equilibrium strategy profiles for 3-player games. Furthermore, Nash equilibria only guarantee that a single player cannot profit by changing

their strategy, so the possibility of the other two players deviating makes the notion of optimal play more complicated in the first place. Despite the lack of theoretical guarantees, CFR has been shown to perform well in practice in small 3 player games [SGS13].

In practice, the size of trick taking card games poses a significant issue. DeepStack uses counterfactual value predictions made for each possible holding for the opponent. In Texas Hold'Em, player hands contain only 2 private cards — yielding $\binom{52}{2} = 1,326$ possible holdings. Many of these holdings can then be grouped together effectively using domain knowledge. In trick-taking card games, player hands typically contain many more cards. In Skat for instance, there are $\binom{32}{10} = 64,512,240$ possible hands per player. The game's branching factor further complicates matters. At the beginning of the cardplay phase, there can be up to 2.04×10^{25} terminal histories leading from each information set; this is several orders of magnitude larger than any previous successful application of CFR [Mor+17], and thus likely too large for sample-based CFR variants such as Outcome Sampling Monte-Carlo CFR [Lan+09].

Finding a good abstraction for Skat and other trick-taking card games may be the key to successfully applying CFR but is challenging for its own reasons. Grouping cards or hands together in trick-taking card games is difficult because the rules of the game dictate that the presence of a single card in one opponent's hand instead of the other's can completely change the result of the game. Learning abstractions with Regression CFR [Wau+15] is a possibility but is also challenging because of the sheer number and size of information sets. Determining which actions generalize across which information sets is complex because of the aforementioned effect of single cards. Differing sets of legal actions across information sets also poses a challenge.

2.6 Determinization, Evaluation, and Inference

So far, previous work has been unable to overcome the challenges discussed in the previous section required for applying CFR to trick-taking card games.

Instead, the state of the art consists of less theoretically-sound algorithms for approximating action values. These algorithms are typically online (the strategy is computed at runtime rather than pre-computed and stored on disk) and have low time and space requirements; their performance is good in practice because they scale to the size of many common trick-taking card games.

To approximate action values in large imperfect information games, these algorithms start by sampling states from the information set. The sampled states are subsequently used by an evaluation component to approximate move values for the information set. This process is known as determinization.

Determinized techniques do not use reach probabilities of terminal histories to calculate Monte Carlo estimates of values. Instead, they use an evaluation component that, given a determinized state, produces a terminal history consistent with that state and returns its utility. This allows estimates to be computed online with low time and space requirements.

Despite limited performance guarantees, determinization can work well in practice and is considered the state-of-the-art in some domains [FB13]. Good performance requires sampling states according to the true distribution of states in the information set: $\eta(s|I)$. If states are not chosen according to $\eta(s|I)$, significant error can be introduced to the resulting action values — regardless of evaluation accuracy. In this work, inference is defined as a tool designed to help approximate $\eta(s|I)$ so that estimated action values in I are as accurate as possible.

Chapter 3

Determinized Evaluation Techniques

Current state-of-the-art algorithms for trick-taking card games rely on determinization. In this process, perfect information states consistent with the observed information are sampled from the player's information set and then evaluated. This approach is theoretically-flawed, but remains relevant because of its undeniable success in certain domains. Much of the recent work in this area has focused on lessening the impact of these flaws and improving the accuracy of state evaluations. This chapter provides an in-depth review on these evaluation techniques.

3.1   Perfect Information Monte-Carlo

Perfect Information Monte Carlo (PIMC) search [Lev89] has been successfully applied to popular trick-taking card games like Contract Bridge [Gin01], Skat [Bur+09], Hearts and Spades [Stu08; SW06]. PIMC is a straightforward technique for computing estimates of action values in an information set.

At its core, PIMC determinizes a state that is consistent with the history of the game and the player to move's private information. Every action available to the player is then evaluated by making the move and then solving a perfect information variant of the rest of the game with a standard search algorithm like minimax. This process is repeated n times, with values for each action summed over each iteration.

```
PIMC(InfoSet I, int n)
    for a ∈ A(I) do
    │   v[a] = 0
    end
    for j ∈ {1..n} do
    │   s ← Sample(I)
    │   for a ∈ A(I) do
    │   │   v[a] ← v[a] + PerfectInfoVal(s, a)
    │   end
    end
    return argmax_a v[a]
```

Algorithm 1: PIMC search.

Game-specific rules are used to sample states consistent with the information set. In trick-taking card games for instance, some candidate states can be eliminated through reasoning about each player's void suits. In the basic algorithm, consistent states are drawn at uniform random, but many practical implementations try to sample from a better estimate of $\eta(s|I)$. Algorithm 1 shows a generic implementation of PIMC.

3.1.1 Criticisms

PIMC has been heavily criticized over the years because it doesn't capture important imperfect information elements of the game tree. Russell and Norvig [RN16] call PIMC "averaging over clairvoyance" because the approach makes the full state observable to all players. This leads to forward search using actions that would not normally be taken under uncertainty. Frank and Basin [FB98] highlight two key problems with PIMC: strategy fusion and non-locality.

Strategy fusion occurs when a player reasons as if they can use a strategy that makes state-specific decisions in every state — regardless of information set structure. Recall that the key distinction between perfect information and imperfect information games in the extensive form is the presence of non-trivial information sets $I \in \mathcal{I}$. Two states $h, j \in I$ are indistinguishable to the player who is choosing an action in I. Allowing player i to use a strategy π_i where

$\pi_i(h) \neq \pi_i(j)$ during simulation leads to inaccurate action value estimates in the real game. Thus, strategies computed with this technique will never take actions related to bluffing, gaining or hiding information from opponents, or sending information to teammates — all of which are considered hallmarks of imperfect information gameplay.

The second issue identified by Frank and Basin, non-locality, stems from the property that a node's value is computed solely based on the values of its children. This fact is used by the perfect information search component of PIMC to compute the action values for each determinized state. However, this structure is not actually present in imperfect information game trees. Values of nodes in imperfect information game trees can depend on other parts of the tree.

Figure 3.1 shows a contrived example of a non-local dependency. This two-player game starts with a coin flip that is observed by Player 1 only. After observing the result of the flip, Player 1 can choose to "play" or "pass". Choosing to pass gives Player 1 an automatic victory if the result of the initial flip was tails. Player 2 cannot observe the result of the coin flip because, if Player 1 decides to play the game, their job is to guess it. However, Player 2 should reason that Player 1 would never choose to play if the result of the flip was tails — making a guess of heads the optimal action in this information set. PIMC, however, will return equal action values for H and T.

3.1.2 Success

Despite all of the criticism, PIMC has remained relevant because it is still among the state-of-the-art algorithms for some games. It's easy to implement and understand, it can give reasonable estimates quickly, and it scales to relatively large games like Skat. Furthermore, it seems that strategy fusion and non-locality only cause severe errors in PIMC under certain conditions — under others, PIMC performs well. In Long et al. [Lon+10] the authors seek to understand under which conditions PIMC succeeds and fails. They show empirical evidence which suggests that for some classes of games, including
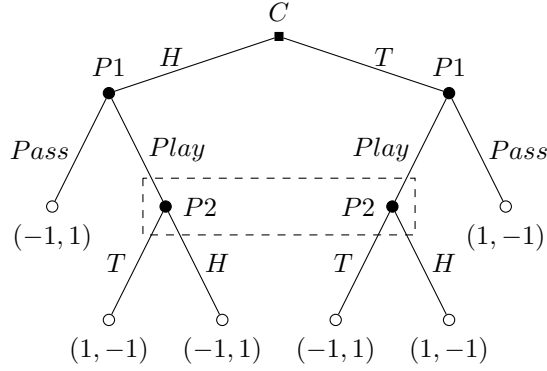
Figure 3.1: An example of non-locality. C is a chance node where a coin is flipped. If the result is heads (H) the game proceeds in the left half of the tree. Although $P2$ can not distinguish between nodes in their information set, $P2$ can determine that the best action is always to guess H because if the initial flip revealed tails, $P1$ would have chosen to pass.

trick-taking card games, "PIMC will not suffer large losses in comparison to a game-theoretic solution."

Long et al. study the effects of 3 different properties on the performance of PIMC against a game-theoretic solution: leaf correlation, disambiguation factor, and bias. Leaf correlation measures how likely sibling terminal nodes are to have the same utility. It describes how much a player can affect their payoff in the final actions of the game. Disambiguation factor reveals the rate at which information set sizes shrink as actions are taken and more information becomes public. Bias describes how often parts of the game tree favor one player over another.

Their findings suggest that games with a relatively high disambiguation factor are favorable for PIMC; trick-taking card games fit this category. This may be because, as more information is revealed, the game becomes closer to a perfect information game. Leaf correlation also has a significant effect on PIMC. When leaf correlation is low, PIMC performs poorly; when it is high, PIMC performs well. Bias, on the other hand, was found to have a minimal effect on the performance of PIMC. The authors go on to explain that trick-taking card games exhibit high leaf correlation and a moderately high disambiguation factor — explaining PIMC's historical success in Skat

19

and Bridge.

3.1.3 Inference in PIMC

Ginsberg's bridge-playing GIB [Gin01] was the first successful application of PIMC in a trick-taking card game. GIB appears to perform some state inference in that it samples "a set D of deals consistent with both the bidding and play", but details regarding the inference are absent from the paper.

Kermit [Bur+09; FB13] uses a table-based procedure that takes opponent bids or declarations into account to infer the likelihood of states within an information set. Unlike this work, Kermit does not use the sequence of actions during the cardplay phase for further inference — only marginalizing over its own private cards and those that have already been played.

3.2 Information Set Monte Carlo Tree Search

Another category of methods based on Monte Carlo Tree Search (MCTS) keep statistics about action values and visit counts at interior nodes of the game tree. They use these values to select which actions should be taken to advance the search. This is the main distinction from PIMC, where action values are only kept at the root node and the search is advanced by solving for the minimax value.

Combining MCTS methods with Deep Learning [LBH15] led to groundbreaking results in perfect information games. In particular, AlphaGo Zero far surpassed human-level play in Go, Chess, and Shogi [Sil+16; Sil+17a; Sil+17b] — a feat previously considered many years away for AI.

Like PIMC, MCTS is an anytime algorithm; value estimates can be computed given any amount of time available. Naturally, estimates typically become more accurate given more time. MCTS has been traditionally used in only perfect information games, but extensions have been designed for games with imperfect information as well.

3.2.1 UCT

The upper confidence bound for trees (UCT) [KS06] was the catalyst for a series of successful MCTS applications. It treats each node in the game tree as a multi-armed bandit problem — calculating the node's value using the UCB1 algorithm [ACF02] shown in Equation 3.1.

$$V_j = \bar{X}_j + c\sqrt{\frac{\ln n}{n_j}} \tag{3.1}$$

$\bar{X}_j$ is the current estimate of the expected value of node j. n is the number of times j's parent has been been visited, and n_j is the number of times that j has been selected from its parent. c is a scalar that must be tuned to reach a problem-specific optimal value. During forward simulation, the maximum-valued child node $argmax_j V_j$ is chosen to advance the state of the game. The main idea of UCB1 is that value estimates of infrequently taken actions are inaccurate. The choice of c determines how aggressively the algorithm will "explore" actions before trusting these estimates.

Cowling, Powley, and Whitehouse [CPW12] call trick-taking card games subset-armed bandit problems. This means that the set of legal actions is not the same at every node, so issues like over-exploring rare actions can be an issue. The authors suggest either replacing n with the number of times the parent was reached and action j was available, or keeping separate statistics for each possible subset of actions in an information set.

3.2.2 Determinized UCT

Determinized UCT is the most straightforward extension of MCTS to imperfect information games. As with other determinized approaches, it starts by sampling a state from the root information set. This state becomes the ground truth for forward simulation and each player's legal moves are determined by it. Actions are chosen according to UCB1 (modified for subset-armed bandits), and statistics are kept for every node (fully-observable state) in the tree. The algorithm returns the action taken the most during simulation as its estimate for the best move.

This algorithm is only slightly different than PIMC and although it does not use minimax, it should converge to the same result after enough simulations. Determinized UCT suffers from the same theoretical issues with strategy fusion and non-locality as PIMC because move statistics are kept at fully-observable nodes.

Determinized UCT was applied to Skat [SBH08], but its performance falls short of PIMC players. Sturtevant [Stu08] also applied UCT to Hearts — yielding a new state-of-the-art at the time.

3.2.3 SO- and MO- ISMCTS

Single-Observer Information Set MCTS (SO-ISMCTS) [CPW12] is an attempt to reduce errors caused by strategy fusion by keeping statistics at the information set level. That is, nodes in the tree on a given iteration are information sets consistent with a determinized state for that iteration rather than fully determinized states. This means value estimates are expectations over all possible states in an information set. The idea is that this prevents the algorithm from cherry-picking actions for individual states.

Multi-Observer ISMCTS (MO-ISMCTS) extends SO-ISMCTS to handle games with partially observable moves. The algorithm maintains a tree for each player in the game, with edges only occurring when that specific player chooses an action. The trees are descended simultaneously, and the algorithm proceeds in the same fashion as SO-ISMCTS.

Furtak and Buro [FB13] argue that ISMCTS introduces new issues of its own while trying to reduce errors from strategy fusion. For instance, the player leaks private information to their playout adversaries by only sampling states consistent with the player's private information. Allowing the strategies of playout adversaries to adapt across roll-outs biases the action value estimates with this information. The search returns the move values assuming the opponent will react perfectly to your hand — leading to the same fundamental issues as PIMC. One suggested remedy is sampling both consistent and inconsistent states, but this makes the search space intractable for many

applications.

3.2.4 Inference in ISMCTS

As with other determinized techniques, variants of ISMCTS can be improved using inference to estimate the state probability distribution and sample more realistic states from information sets.

Baier et al. [Bai+18] use a different type of inference to bias MCTS results. They propose a method for biasing MCTS that boosts the scores of nodes reached by following actions judged likely to be played by humans according to a supervised model. This focuses the algorithm on certain states in the game tree that are deemed more likely, given that the player is playing with humans. Playing like a human can have other advantages as well; in games where cooperation is required, a player that uses human conventions may fare better than one that does not. However, this approach still assumes that $\eta(s|I)$ is a uniform random distribution, so the resulting action values are susceptible to error because unlikely states may be determinized in the first place.

3.3 Imperfect Information Monte-Carlo

Furtak and Buro [FB13] take a different approach to avoiding strategy fusion errors and implement a recursive variant of PIMC — resulting in the current state-of-the-art player for Skat.

Imperfect Information Monte Carlo (IIMC), seen in Algorithm 2, consists of a top-level player that uses of a lower-level player to finish games for action value estimates. The top-level player samples a state from the current information set and then specifies the first action that will be taken by the lower-level player. The lower-level player takes the action and then proceeds to finish the game (against copies of itself) given the underlying state that was previously sampled. The lower-level player's policy must be fixed, or IIMC has the same information-leaking problem as ISMCTS. The utility obtained at the terminal state is returned to the top-level player, where it is used to estimate the value

```
IIMC(InfoSet I, int n, Player p)
    for a ∈ A(I) do
        │ v[a] = 0
    end
    for i ∈ {1..n} do
        │ s ← Sample(I)
        │ for a ∈ A(I) do
        │ │ v[a] ← v[a] + FinishedGameValue(s, a, p)
        │ end
    end
    return argmax_a v[a]

FinishedGameValue(State s, Action a, Player p)
    s = MakeMove(s, a)
    while s is non-terminal do
        │ s = MakeMove(s, ComputeMove(p, ToInfoSet(s)))
    end
    return u(s)
```

Algorithm 2: IIMC search.

of the initial action. This process is repeated for every action $a \in A(I)$ using n sampled states from I.

IIMC partially resolves strategy fusion by forcing the player to simulate the first action at the information set level, and then advancing the state before the lower-level player plays out the rest of the game. However, if the lower-level player uses an algorithm like PIMC, then the same theoretical issues bias the utilities returned to the top-level.

IIMC's playing strength and decision-making speed are greatly influenced by which lower-level player is used. Using a PIMC lower-level player (RecPIMC) was shown to select actions 20× slower than a standard PIMC player in Skat, but achieved a new state-of-the-art in playing strength [FB13]. Faster, weaker lower-level players like XSkat[1] can be used. Although this leads to much stronger play than regular XSkat, it was shown to perform significantly worse compared to using a PIMC lower-level player [FB13].

---

[1]XSkat is a free software Skat program written by Gunter Gerhardt (www.xskat.de). It uses a large set of hand-designed rules and can play games very quickly.

3.3.1 Inference in IIMC

Furtak and Buro [FB13] use bidding and declaration information for inference in both the top-level and the lower-level player in their RecPIMC implementations for Skat. However, this idea could be extended further by incorporating action history from the cardplay phase to make the estimates of $\eta(s|I)$ more accurate.

Since RecPIMC significantly outperformed IIMC with weaker lower-level players like XSkat, it is reasonable to assume that further improving the lower-level player will lead to another increase in performance. Better top-level sampling should also be beneficial to performance if it can lead to more realistic states passed to lower-level players for evaluation.

3.4 Inference and Better Determinization

These notable contributions improve state evaluation quality, but fail to adequately capture the effect of action history on information set state distributions. As previously mentioned, a bad estimate of $\eta(s|I)$ leads to inaccurate action values. The authors of these contributions have identified this issue, but little action has been taken to improve inference for determinized evaluation techniques.

Cowling, Powley, and Whitehouse [CPW12] say that inference is essential for optimal play and that it helps reduce errors that are the result of non-locality. Furtak and Buro [FB13] also mention that lack of an inference module is hurting their results as a result of non-locality. In the remainder of this thesis, I present and validate a novel approach for improving inference in this domain.

Chapter 4

Estimating State Probabilities

Previous work in trick-taking card games uses table-based methods for inference. This works well if the state representation is small enough that there is sufficient data corresponding to every table entry. However, as representations grow larger and the amount of training data for each representation declines, table-based approaches become more prone to overfitting and more difficult to work with in practice. Excluding or abstracting parts of the state is an option, but doing so may sacrifice predictive power. An approach that is able to generalize across similar states with a sufficient, yet reasonable, amount of data allows for more state information to be considered, which results in better predictions.

In trick-taking card games like Skat, card history has a significant impact on $\eta(s|I)$. Ignoring it for inference is unthinkable to even a beginner human player, but it has not been incorporated in previous work because it makes the state too large for table-based techniques. This chapter presents a novel technique for inferring $\eta(s|I)$ using card history in trick-taking card games.

4.1 Individual Card Distributions

Deep Learning [LBH15] can be used to learn complex representations of the input feature space suitable for certain classification tasks. Models can be trained to approximate functions with large input feature sets — large enough to represent the full state in a game like Skat. However, learning to predict the

correct state directly is not feasible when information sets contain potentially millions of states. This section describes a technique that estimates probabilities for the locations of individual cards and uses that to estimate $\eta(s|I)$ instead.

4.1.1 Estimating Card Location Distributions with Deep Neural Networks

Given input features describing information set I, we can independently predict the location of each individual card $c \in C$ by modelling it as a multi-class classification task. Given a feature set $\phi(I)$, $L(\phi(I))$ is a real $|C| \times l$ matrix, where l is the number of possible card locations (e.g., $|C| = 32$ and $l = 4$ in Skat because a card is located either in hand 1, 2, 3, or the skat). Each row $L(\phi(I))_c$ is a probability distribution over the possible locations of card c.

Input features provide the classifier with obvious information like which cards have already been revealed or are in the player's hand, but also with more subtle information like which suits opponents have revealed as void. The goal of the classifier is to uncover subtleties and interactions between input features that can help reveal likely locations for the remaining of the cards.

4.1.2 Computing State Probabilities

If individual card predictions are accurate, they can be used to estimate $\eta(s|I)$. Formally, individual card location probabilities are used to predict state probabilities by applying Equation 4.1. By assuming independence and multiplying the probabilities of each card c's true location in state $s \in I$, I provide a measure for each state that can be normalized into a probability distribution for states in the information set:

$$\eta(s|I) \propto \prod_{c \in C} L(\phi(I))_{c,loc(c,s)} \tag{4.1}$$

Here $loc(c, s)$ is the true location of card c in state s.

This process is computationally expensive in early tricks where the information sets are relatively large. However, only a single forward pass is

performed per information set, so the performance bottleneck is multiplying the card probabilities for each state and normalizing the distribution.

4.2 Application to Skat

Disregarding some notable exceptions, modern learning techniques tend to benefit from good hand-engineered features, and in the case of neural networks, considerable network design efforts. This section describes a successful approach to feature engineering and network design in Skat, but it can be adapted to other games using domain-specific knowledge.

4.2.1 Network Design

Figure 4.1 details the network architecture. Predictions are always made in view of the player to move. I train a separate network for each game type (suit, grand, null). Regardless of the game type, there are 32 total cards in Skat that can be in any of 4 potential locations (3 hands and the skat). Each network has the same overall structure. I use dropout [Sri+14] of 0.8 on layers 2, 3, and 4 and early-stopping [Pre98] on a validation set to reduce overfitting. Table 4.1 lists all hyperparameters used during training. Hidden layers use ELU activations [CUH15], and softmax is applied to each individual card output.

One practical insight for training this type of classifier is that learning is simpler when the full targets are used, rather than attempting to predict only the unknown elements. In Skat for example, this means predicting the full 32-card configuration from the beginning of the cardplay phase.

Adapting individual card inference to other trick-taking card games simply requires training a neural network with game-specific input features and an appropriate output size. As explained in Equation 4.1, network output size must be defined by the number of cards $|C|$ and the number of possible locations for each card l according to the rules of the game.
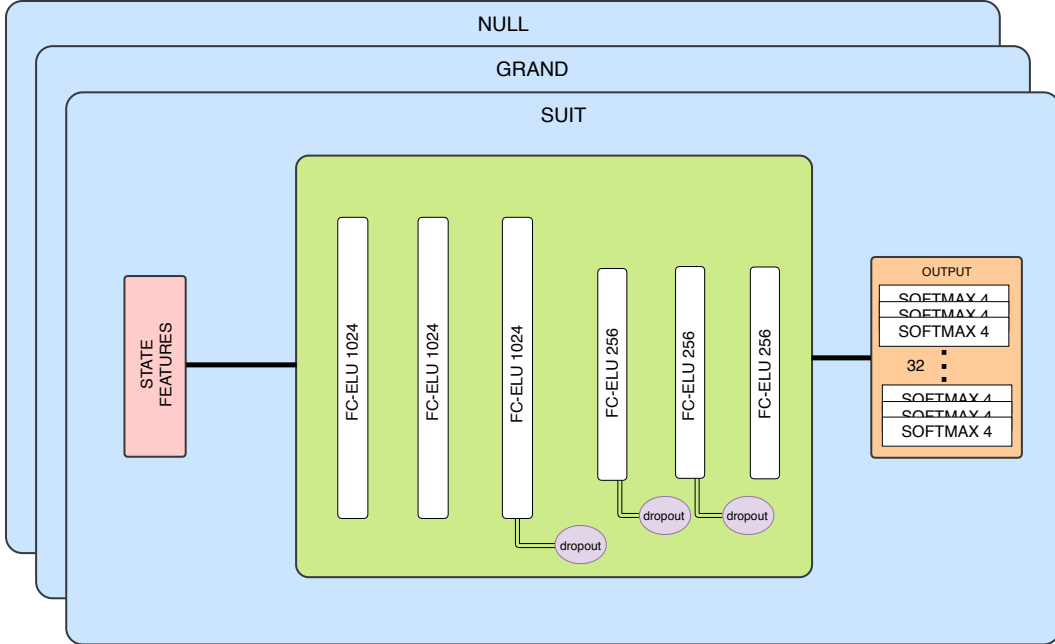
Figure 4.1: Inference network architecture.

Table 4.1: Inference network training hyper-parameters.

Parameter	Value
Dropout	0.8
Batch Size	32
Optimizer	ADAM
Learning Rate (LR)	10^{-4}
LR Exponential Decay	$0.96/10^7$ batches

4.2.2 Feature Engineering

Various input features, listed in Table 4.2, are used to represent the information set of the player to move. Lead cards are the first cards played in a trick, and sloughed cards are those that are played when a player cannot follow suit but also does not trump. Void suits indicate when players' actions have shown they cannot possibly have a suit in their hand. Bidding features are broken down into type and magnitude. Type indicates a guess as to which game type the opponent intended to play had they won the bidding with their highest bid. This is computed by checking if the bid is a multiple

29

Table 4.2: Inference network input features.

Feature	Width
Player Hand	32
Skat	32
Played Cards (Player, Opponent 1&2)	32*3
Lead Cards (Opponent 1&2)	32*2
Sloughed Cards (Opponent 1&2)	32*2
Void Suits (Opponent 1&2)	5*2
Max Bid Type (Opponent 1&2)	6*2
Max Bid Magnitude (Opponent 1&2)	5*2
Current Trick	32
Soloist	3
Trump Suit	5
Total	360

of each game type's base value. Magnitude buckets the bid into 1 of 5 ranges that are intended to capture which hand multiplier the opponent may possess. Domain knowledge is used to construct ranges that group different base game values with the same multiplier together. The exact ranges used are $18..24, 27..36, 40..48, 50..72,$ and > 72. These ranges contain some unavoidable ambiguity because some bids are divisible by multiple game values, but bid multiplier is a strong predictor for the locations of the jacks in particular. The soloist and trump suit features indicate which player is the soloist and which suit is trump for the current game, respectively. All features are one-hot encoded. Network output is used as described in Equation 4.1 to compute probabilities for individual states and build an estimate of $\eta(s|I)$.

4.2.3 Training

Networks are trained by minimizing the average cross-entropy across individual card outputs on a training set of example states. Cross-entropy, H, between

two discrete probability distributions p and q is defined as:

$$H(p, q) = - \sum_x p(x) \log q(x) \qquad (4.2)$$

If H is the objective function used for training, p is derived from the underlying perfect information state of an example information set, and it has zero probability mass everywhere except the true location of each card $c \in C$ in that state. This means that only one term in H is non-zero for every card. q is the softmax distribution output by the network given $\phi(I)$. Averaging H over all cards yields *CE*.

$$CE = - \mathbb{E}_{c \in C}[\log L(\phi(I))_{c, loc(c,s)}] \qquad (4.3)$$

Training data consists of example states, randomly sampled at a rate of 1/3, from approximately 20 million games played by humans on a popular Skat server [DOS18]. Omitting some states is necessary for improving generalization because states from the same game are highly correlated. The data is extracted from only the first 9 tricks because the last trick has only one possible action and is therefore not a decision point. Training and validation set sizes are listed in Table 4.3. Set sizes are imbalanced between game types because some games are played more frequently than others, but balancing them is unnecessary because a separate network is trained for each game type.

Networks are trained for a maximum of 10 epochs over their corresponding training sets, but early-stopping activates earlier in each case. This may indicate that the datasets are larger than necessary. Each epoch takes approximately 10 hours for the largest dataset (suit games) — meaning the training process is relatively slow. This is mainly because the datasets are several hundred gigabytes uncompressed, so batches need to be uncompressed on demand. The training process uses Python Tensorflow [Aba+16].

4.3 Card Prediction Performance

This section describes how I evaluate the performance of individual card inference models. It includes the definition of some baselines and a description

Table 4.3: Training and validation set sizes (in millions of states) for each game type.

Game Type	Grand	Suit	Null
Training	24.6	66.3	2.2
Validation	5.8	14.3	0.5

of two metrics used for evaluation. I will refer to the deep neural network described in Section 4.2 as **Deep** from this point forward.

4.3.1 Baselines

The first baseline, **Uniform**, can be considered as a lower-bound on inference performance. After accounting for all played cards, all cards in the player to move's hand, the skat (if known), and any opponent void suits, it treats all unknown cards as uniformly distributed between possible locations. Individual card distributions are computed from the normalized frequencies of each card's location over all states in the player to move's information set.

Kermit uses the table-based inference technique employed by Kermit's SD version ("Soloist/defender inference") described in [Bur+09]. It estimates $\eta(s|I)$ given the limited context of actions taken in the bidding, discard, and declaration phases of Skat.

Logistic uses softmax regression on the feature set described above to make predictions about the locations of individual cards. It is implemented in Tensorflow as a densely-connected network with no hidden layers. This baseline is included to help determine if a deep learning approach is warranted for this task. Logistic uses the same feature set as Deep; all input features are one-hot encoded and thus already prepared for this type of learning. With 360 input features and $32 \times 4 = 128$ outputs, this model contains 46,080 total weights — far less than the 1.84 million weights used in Deep. Moreover, this model is simpler, more interpretable, easier to train, and faster to use, so Deep must provide a substantial performance boost to be considered justifiable.

4.3.2 Average Cross-Entropy of Card Predictions

The first evaluation metric I use to measure prediction performance comes from the objective function used for training: *CE*. As in training, lower *CE* is better. It indicates that more information has either been revealed or inferred by the model. *CE* will naturally start higher and decrease rapidly as more cards are played because $\log L(\phi(I))_{c,loc(c,s)}$ should be close to 0 when $loc(c,s)$ is already known. Therefore, I calculate and report *CE* independently for every possible number of cards known to the player to move during the cardplay phase.

Figure 4.2 shows *CE* of output distributions from Uniform, Logistic, and Deep calculated on an independent test set. The results are split by game type and whether or not the player knows the contents of the skat.

Not knowing the contents of the skat increases uncertainty because there is an additional potential location for unknown cards. When the skat is not known, Deep and Logistic's inference capabilities outperform Uniform. However, when the skat is known, it seems that the inference capabilities of both models is limited in comparison. This could be because any opponent void suit indicates the exact location of all remaining unknown cards in that suit; they must be in the other opponent's hand if the skat is known. This neutralizes inference capabilities because much of the information that could be inferred is instead directly revealed to the player. Nevertheless, both models still outperform the baseline over most of the game in all game types.

Logistic performs relatively well compared to Uniform and Deep during the early stages of the game. Part of this performance can be explained by the quality of the engineered feature set. That being said, the effect of representation learning is shown by the overall superior performance of Deep. Logistic's performance rapidly deteriorates toward the end of the game when the model must account for complex interactions between the input features. Learned representations of these features seem to be providing Deep with additional predictive power in this case.

The difference between Deep and Logistic is smaller in null games. This may be due to the relatively small training set size for this game type. Null is
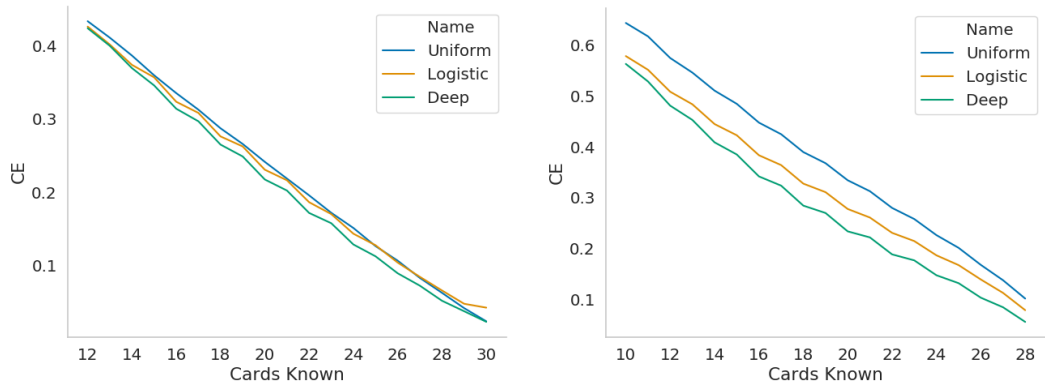
(a) Grand
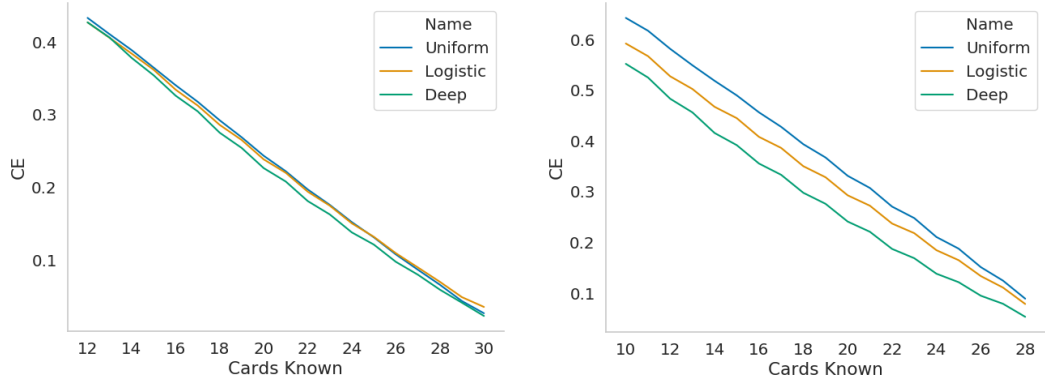
(b) Suit
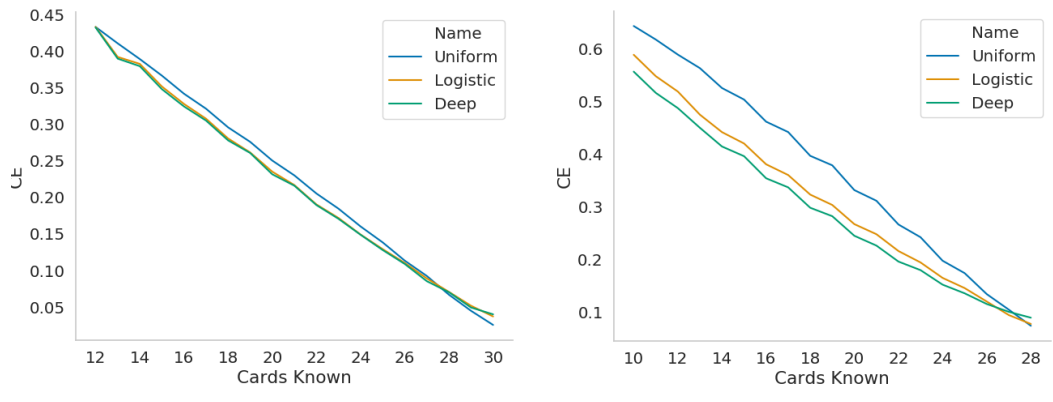
(c) Null

Figure 4.2: Average *CE*, split by game type, number of unknown cards, and whether the player knows the skat (left) or not (right).

the rarest game type and the outcome of null games is often decided before all cards have been played — making a sufficient amount of training data for a deep neural network difficult to obtain. The logistic model generally requires less data and is easier to train effectively, but fails in comparison to the more complex deep model when there is sufficient data.

A larger *CE* than Uniform is observed in some instances. This indicates that a model is not accounting for cards that have been revealed, or is making predictions that are consistently worse than randomly assigning the locations of the remaining unknown cards. For Deep, this only occurs at the end of null games when almost all cards have been revealed. Possible reasons for this include the aforementioned lack of training data for null games and the fact that the models are not trained on example states from the final trick of the game.

4.3.3 True State Sampling Ratio

To test how well combining the location probabilities of individual cards approximates $\eta(s|I)$, I compare the probability of sampling the true state from the estimated distribution and the probability of uniformly randomly sampling it from the information set. This comparison provides the True State Sampling Ratio (*TSSR*) which conveys how many times more likely the true state will be selected, compared to uniform random sampling.

$$TSSR = \frac{\eta(s^*|I)}{1/|I|} = \eta(s^*|I) \cdot |I| \tag{4.4}$$

$\eta(s^*|I)$ is the probability the true state is sampled in the estimated distribution, and $|I|$ is the number of states in the current information set. Sampling the true state with a higher probability should lead to performance improvements in determinized evaluation techniques.

TSSR is calculated for each baseline and each trick, with defender and soloist and game types separated. Like *CE*, *TSSR* is evaluated using only the first 9 tricks. Each card number (0 through 26), role (defender or soloist), and game type were evaluated for each algorithm on samples from 3,000 games

from holdout sets of games previously played by humans.

Figure 4.3 shows the average value of *TSSR* for each model as well as a strict upper bound for *TSSR* dubbed the **Oracle**. The Oracle predicts the true world's probability to be 1.0, so its *TSSR* value is equivalent to $|I|$. The value of *TSSR* is markedly impacted by both game type and player role.

For all models, *TSSR* is larger for defender compared to soloist. This is due to the declarer choosing the game that fits their cards, making inference much easier for the defender. Furthermore, soloists know the locations of more cards to begin with because they know the skat — meaning that there is less potential for inference in the first place. As more private information is revealed throughout the game, *TSSR* decreases exponentially along with the number of states per information set. This rapidly increases the probability of selecting the true state with a random guess and equalizes the inference capabilities of all algorithms by the end of the game.

For Logistic and Kermit, *TSSR* reaches its peak in early tricks and decreases over time. Deep, however, peaks around tricks 3-5, and performs consistently better in the middle of the game. This can be attributed to the successful inclusion of more state information as input for prediction. With the exception of the very first trick for grand and suit defender games, Deep performs considerably better compared to all other inference techniques tested.

In terms of the likelihood of sampling the true state, Deep is the strongest of all algorithms considered. The other two algorithms perform similarly well, with Logistic having the edge as soloist, and Kermit having the edge as defender. This is likely due to how Kermit's inference leverages the additional information of the soloist's game declaration when on defense. Furthermore, Kermit's poor performance as the soloist suggests that defender bidding information is far from sufficient for good inference as the soloist. These results clearly show the benefit of using card history for inference, and suggest that learned representations of the input features are providing a considerable boost to inference performance.
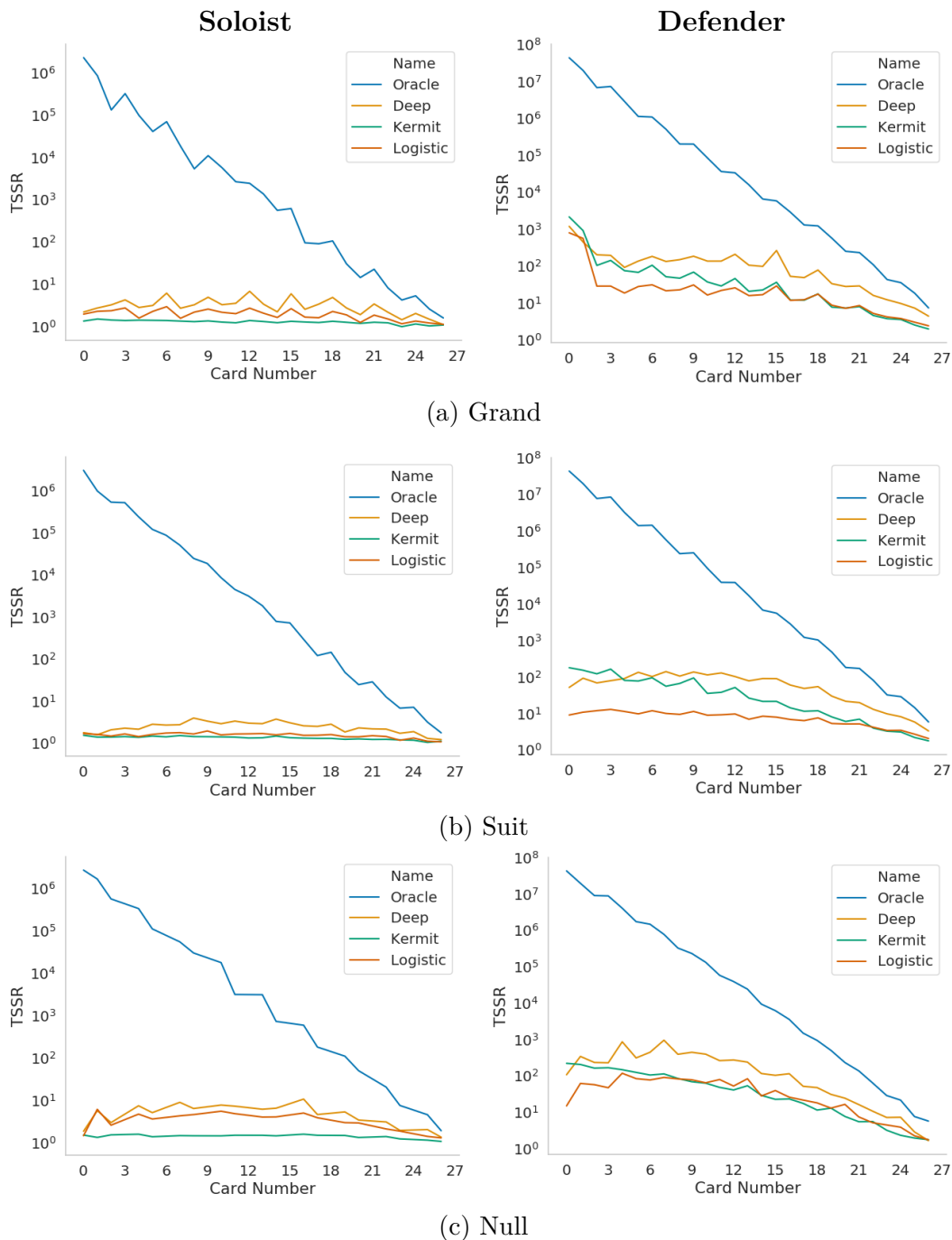
Figure 4.3: Average *TSSR* after *Card Number* cards have been player for Deep, Logistic, Kermit, and Oracle inference. Data is separated by game type and whether the player to move is the soloist (left) or a defender (right).

Chapter 5

Combining Inference With Search

The previous chapter demonstrates a technique for performing inference based on action history in trick-taking card games. However, the technique I presented is more compelling if it can be used to improve the performance of AI players. In this chapter, I show how to apply cardplay inference to improve determinized search in the application domain of Skat, and I test the tournament performance of several different inference techniques in head-to-head matchups.

5.1   Improving Determinized Evaluation

This section lists the modifications needed to supplement existing determinized evaluation techniques with individual card inference. In general, the approach consists of estimating $\eta(s|I)$ at each information set independently. This requires a single forward pass of the network to compute $L(I)$, followed by a series of multiplications to compute Equation 4.1. The result must be normalized into a probability distribution over $s \in I$ before states can be sampled.

5.1.1   PIMC+

Algorithm 3 shows the modifications necessary to estimate $\eta(s|I)$ using individual card inference and apply the result to PIMC. `ProbabilityEstimate` implements Equation 4.1. I call this specific method of biasing sampling `PIMC+`

```
PIMC+(InfoSet I, int n)
    for a ∈ A(I) do
    │   v[a] = 0
    end
    for s ∈ I do
    │   p[s] ← ProbabilityEstimate(s, I)
    end
    p ← Normalize(p)
    for i ∈ {1..n} do
    │   s ← Sample(I, p)
    │   for a ∈ A(I) do
    │   │   v[a] ← v[a] + PerfectInfoVal(s, a)
    │   end
    end
    return argmax_a v[a]
```

Algorithm 3: PIMC with individual card inference.

to differentiate from previous inference techniques; biasing sampling in PIMC is not a new idea in itself.

Although this approach requires extra computation every time the player is to move, this subtle modification to PIMC has the potential to significantly improve playing strength by forcing the algorithm to evaluate more likely states. Furthermore, it does not increase decision-making time beyond what is considered reasonable by humans. The maximum number of states in an information set in Skat (approximately 42 million) is manageable in approximately 2 seconds per move on a single core of an Intel® Core™ i7-8700K and our current implementation could easily be parallelized.

5.1.2 RecPIMC+

Applying the same idea to IIMC, or Recursive PIMC in particular, is straight-forward. Algorithm 4 shows how individual card inference can be applied in both the top-level and lower-level players.

RecPIMC is the state-of-the-art algorithm for Skat, but the amount of time it takes per decision makes it too slow for play with humans if the algorithm is running on commodity hardware. In Section 5.2.4, I show that incorporating

```
RecPIMC+(InfoSet I, int n, int m)
    for a ∈ A(I) do
     |  v[a] = 0
    end
    for s ∈ I do
     |  p[s] ← ProbabilityEstimate(s, I)
    end
    for i ∈ {1..n} do
     |  s ← Sample(I)
     |  for a ∈ A(I) do
     |   |  v[a] ← v[a] + FinishedGameValue(s, a, m)
     |  end
    end
    return argmax_a v[a]

FinishedGameValue(State s, Action a, int n)
    s = MakeMove(s, a)
    while s is non-terminal do
     |  s = MakeMove(s, PIMC+(ToInfoSet(s), n))
    end
    return u(s)
```

Algorithm 4: Recursive PIMC with individual card inference.

individual card inference into RecPIMC and optimizing sampling leads to a new state-of-the-art for Skat cardplay tournament performance.

5.2 Experiments

I measure the effect of using inference in determinized search-based card players using each of the baselines explained in Section 4.3.1. Performance is measured using by playing Skat tournaments and observing the resulting scores under the Fabian-Seeger rule set.

Tournaments are structured so that pairwise comparisons can be made between players. Two players play 5,000 matches in each matchup, and each match consists of two games. All matchups use the same set of games for each game type. Each player gets a chance as the soloist against two copies of the other player as defenders. The games start at the cardplay phase — with bidding, discard, and declaration previously performed by human players on

the DOSKV server. These games are from a separate set than those used for training and validation, and I calculate and report results separately for each of Skat's game types.

I test for statistical significance between mean player scores using the Wilcoxon signed-rank test [WKW70] with significance level 0.05. Data are paired because each game is replayed with the soloist and defender roles reversed. The games come from a randomly-selected holdout set, and scores are measured on a ratio scale. Therefore, none of the test's assumptions are violated.

5.2.1 PIMC Cardplay Tournaments

I present results from tournaments where all players use PIMC for evaluation and only vary in how they select states to evaluate in this section. Uniform represents the simplest way to estimate $\eta(s|I)$, so it should be considered as a true baseline for PIMC. Kermit is considered the state-of-the-art for PIMC-based Skat players. In all matchups, one standard deviation is no more than 0.7, 0.55, and 0.53 tournament points per game for suit, grand, and null games respectively.

Table 5.1 shows results from each tournament type. The positive effect of my sampling technique is clearly shown in all game types, with the point difference between Deep and Kermit always statistically significant. Deep's cardplay tournament performance is an exciting result considering that Kermit was already judged as comparable to expert human players [Bur+09]. Additionally, a per-game tournament point increase of more than 3 in suit games and 4 in null games means that the gap between the two players is substantial. To provide some context to the magnitude of this difference, consider that Ron Link, a world champion caliber Skat player, outperforms other expert players at the Edmonton Skat Club by an average of approximately 4.1 tournament points per game [Lin18].

Each player's average time per move is reported in Table 5.2. This data was calculated on a separate set of 1000 games using a single machine to

	Grand			Suit			Null		
	$n = 80$	$n = 160$	$n = 320$	$n = 80$	$n = 160$	$n = 320$	$n = 80$	$n = 160$	$n = 320$
Kermit vs. Uniform									
$P1$	39.72	39.26	39.28	24.05	23.23	22.94	17.30	16.65	17.17
$P2$	37.57	37.77	37.75	17.23	18.00	18.11	10.68	11.09	10.77
Δ	2.15	1.49	1.53	6.82	5.23	4.83	6.62	5.56	6.40
Logistic vs. Uniform									
$P1$	39.61	39.39	39.12	22.61	22.44	22.17	17.10	16.96	16.90
$P2$	37.73	37.45	37.92	20.14	19.39	19.60	10.82	10.68	10.57
Δ	1.88	1.94	1.20	2.47	3.05	2.57	6.28	6,28	6.33
Deep vs. Uniform									
$P1$	40.17	40.01	39.39	24.16	23.87	23.80	18.40	17.89	18.43
$P2$	36.46	35.81	36.46	16.14	16.15	15.98	7.98	8.23	7.72
Δ	3.71	4.20	2.93	8.02	7.72	7.82	10.42	9.66	10.71
Logistic vs. Kermit									
$P1$	38.51	38.53	38.08	18.73	18.66	19.11	12.93	13.13	13.23
$P2$	38.47	37.38	38.28	21.87	21.25	20.91	12.71	12.36	12.04
Δ	0.04	1.15	-0.20	-3.14	-2.59	-1.80	0.22	0.77	1.19
Deep vs. Kermit									
$P1$	38.46	38.68	38.84	20.59	20.31	20.73	14.17	14.17	14.31
$P2$	36.92	36.46	36.32	17.59	17.69	17.42	9.87	9.65	9.34
Δ	1.54	2.22	2.52	3.00	2.62	3.31	4.30	4.52	4.97
Deep vs. Logistic									
$P1$	38.66	38.38	38.71	22.50	21.96	21.82	13.92	13.85	13.33
$P2$	36.85	37.05	36.57	16.85	16.81	16.82	10.09	9.83	10.31
Δ	1.81	1.33	2.14	5.65	5.15	5.00	3.83	4.02	3.02

Table 5.1: Tournament points/game for PIMC players. Separated by game type, and number of states evaluated per legal action.

control for hardware differences. Deep's move time statistics suggest that the increase in performance comes at a significant cost; it is roughly $8\times$ slower than Kermit when evaluating 320 states. In practice, 2.43 seconds per action is still within the range that is suitable for play with humans, but games with larger information sets than Skat could pose a problem. Section 5.2.3 shows an approximation technique designed to alleviate this issue.

Uniform's performance reiterates that some type of inference is undoubtedly beneficial to PIMC players in all game types, but the effect of inference in grand games is less noticeable in general. This may be due to a bias introduced from the test set being generated by human players and is discussed further

Table 5.2: Average time (in seconds) taken per action by PIMC players on an Intel® Core™ i7-8700K processor.

Num States	80	160	320
Deep	2.26	2.31	2.42
Logistic	2.24	2.32	2.43
Kermit	0.09	0.17	0.32
Uniform	0.08	0.14	0.27

in Section 5.2.2.

The tournaments between Deep and Logistic suggest that the complex interactions learned between input features help the predictive power of the resulting inference model, which, in turn, causes a substantial increase in playing strength. For suit and null games in particular, it seems that Deep has learned important representations of the input features that the linear model was unable to capture.

Results for matches between Kermit and Logistic show that Kermit's count-based approach may be more robust, in this case, than attempting logistic regression with such a high-dimensional feature space. Logistic seems to perform particularly poorly in suit games — perhaps due to an inability to sufficiently capture the relationships between input features such as which cards have been played and which suit is trump. The Logistic player performs similarly to Kermit in the other game types; however, Kermit's inference does not account for actions made in the cardplay phase. Considering its performance in terms of *CE* and *TSSR*, Logistic performs worse than expected in the tournament setting. This indicates that small performance increases in these metrics can lead to large differences in practical playing strength.

These results suggest that decomposing $\eta(s|I)$ into a product of individual card-location probabilities (Equation 4.1) is a useful approximation. It allows for more state information to be considered when making predictions and the results in a better approximation of $\eta(s|I)$ than shown in previous work. Furthermore, knowing where cards lie allows a search-based player to spend

Table 5.3: Soloist win percentage across all matchups for each game type. *Human* denotes the actual soloist wining percentage from the original games.

Game Type	Grand	Suit	Null
Deep	93.14	81.01	62.93
Logistic	92.63	78.99	61.72
Kermit	92.04	78.52	61.13
Uniform	93.32	77.96	58.99
Human	93.23	80.08	62.62

more of its budget on likely states. From the tournament results, it is clear that this has a positive effect on performance.

5.2.2 Game Type Effects

Tournament results are reported separately for each of Skat's game types because each one has a unique set of rules and a large discrepancy in tournament points earned by winning. Differences in rules mean that, for inference purposes, an action in one game type may have completely different implications than the same action in a different game type.

With the exception of the row labelled "Human", which represents the actual win rate of the games played on the DOSKV server, Table 5.3 shows the overall win rate of each player in the set of all head-to-head matches detailed in Section 5.2.1. This table is not intended for pairwise comparison between players. There are two important insights related to game type effects found in this table: grand games have a higher soloist win percentage and null games seem the most difficult to win consistently.

As seen from the Human player data in Table 5.3, conservative human play may be the main cause of inference seeming less important in grands. In Skat, grand games are worth more than any other game type. They offer a hefty reward when won, but a $2\times$ penalty when lost. If human players are unwilling to take such a risk, then they will only bid on and declare grand games when they are relatively easy to win. This leads to a set of games that

are biased; good players won't benefit as much from superior play. The games are too easy for the soloist and too hard for the defenders for skill to make a difference.

A somewhat related explanation can be made for the surprising difficulty all tested players seem to have playing null games as the soloist. Null games have one of the smallest base values for winning or losing in Skat, and they have no possibility of additional multipliers. So when players gamble on the contents of the skat and bid too high relative to their hand, they will often play null games because they are the cheapest to lose. Winning these is notoriously difficult because the soloist's hand would typically contain cards tailored toward the suit or grand game that they intended to play, whereas a null requires completely different cards.

5.2.3 Scaling to Larger Games

In games with large information sets, estimating probabilities for the entire set of states may become intractable. However, because PIMC is easily parallelizable, information sets must contain hundreds of millions or billions of states before this becomes a problem in practice. In such cases, applying individual card inference is still possible by uniformly sampling a subset of states J without replacement and estimating $\eta(s|I)$ as if $I = J$.

If the subset is small enough, the initial sampling step has the potential to reduce the time taken to estimate $\eta(s|I)$ so that it is negligible compared to the time takes to evaluate the states that are eventually sampled. One drawback is that high probability states could be missed completely — especially if the initial subset is too small.

In Skat, the largest information sets in the cardplay phase contain just over 42 million states. The number of states per information set is highest at the beginning of cardplay and decays sharply as the game progresses and more information is revealed; by the end of third trick there are at most 16 unknown card and only $\binom{16}{7}\binom{9}{2} = 411,840$ states in the largest information sets. Thus, approximating $\eta(s|I)$ can save time while only having an effect

	Grand			Suit			Null		
	$n=80$	$n=160$	$n=320$	$n=80$	$n=160$	$n=320$	$n=80$	$n=160$	$n=320$
Deep500k vs. Deep									
$P1$	37.47	36.88	37.08	18.63	18.83	18.72	11.25	10.75	11.25
$P2$	37.31	37.01	37.11	18.5	18.41	18.19	11.35	11.37	11.34
Δ	0.16	-0.13	-0.03	0.13	0.42	0.53	-0.10	-0.62	-0.09
Deep500k vs. Kermit									
$P1$	38.50	38.75	38.64	20.62	20.48	20.49	14.11	14.14	14.63
$P2$	36.75	36.54	36.55	17.7	17.47	17.65	9.77	9.74	9.01
Δ	1.75	2.21	2.09	2.92	3.01	2.84	4.34	4.40	5.62
Deep100k vs. Deep									
$P1$	37.29	36.74	37.36	18.51	18.85	18.58	11.06	10.95	10.48
$P2$	37.64	37.21	37.02	18.97	18.46	18.55	11.39	11.30	11.71
Δ	-0.35	-0.47	0.34	-0.46	0.39	0.03	-0.33	-0.35	-1.23
Deep100k vs. Kermit									
$P1$	38.35	38.64	38.74	20.60	20.49	20.65	13.97	13.87	14.35
$P2$	37.09	36.87	36.52	17.72	17.55	17.45	10.15	9.86	9.68
Δ	1.26	1.77	2.22	2.88	2.94	3.20	3.82	4.01	4.67
Deep100k vs. Deep500k									
$P1$	37.11	37.07	37.22	18.77	18.30	18.16	10.98	11.38	11.05
$P2$	37.57	37.14	37.19	18.78	18.73	18.71	11.38	11.04	11.24
Δ	-0.46	-0.07	0.03	-0.01	-0.43	-0.55	-0.40	0.34	-0.19
Log500k vs. Logistic									
$P1$	38.58	38.44	38.16	20.73	20.35	20.48	12.59	12.49	12.47
$P2$	38.66	38.64	38.47	20.69	20.72	20.35	13.05	13.10	12.84
Δ	-0.08	-0.20	-0.31	0.04	-0.37	0.13	-0.46	-0.61	-0.37
Log500k vs. Kermit									
$P1$	38.46	38.24	38.13	18.94	18.76	18.89	12.57	13.09	13.09
$P2$	38.39	38.42	38.16	21.45	21.37	21.19	13.22	12.45	12.39
Δ	0.07	-0.18	-0.03	-2.51	-2.61	-2.30	-0.65	0.64	0.70
Log100k vs. Logistic									
$P1$	38.81	38.23	38.03	20.69	20.58	20.42	12.71	12.52	12.36
$P2$	38.24	38.04	38.01	20.76	20.25	20.25	12.66	12.85	12.72
Δ	0.57	0.19	0.02	-0.07	0.33	0.17	0.05	-0.33	-0.36
Log100k vs. Kermit									
$P1$	38.51	38.46	38.28	18.62	18.90	19.05	13.16	12.96	12.68
$P2$	38.15	37.7	38.18	21.83	21.12	20.98	12.23	12.48	12.61
Δ	0.36	0.76	0.10	-3.21	-2.22	-1.93	0.93	0.48	0.07
Log100k vs. Log500k									
$P1$	38.72	38.47	38.33	20.66	20.77	20.46	13.02	13.12	12.72
$P2$	38.38	38.05	38.06	20.71	20.13	20.18	12.56	12.36	12.39
Δ	0.34	0.42	0.27	-0.05	0.64	0.28	0.46	0.76	0.33

Table 5.4: Tournament points/game for subset-sampling PIMC players separated by game type and number of states evaluated.

Table 5.5: Average time (in seconds) taken per action by subset-sampling players on an Intel® Core™ i7-8700K processor.

Num States	80	160	320
Logistic500k	0.16	0.22	0.35
Logistic100k	0.10	0.17	0.30
Deep500k	0.16	0.22	0.35
Deep100k	0.10	0.16	0.30

in the early stages of games. It makes sense to save time estimating $\eta(s|I)$ early in the game because—even with a probability estimate for every state in the information set—sampling the true state as part of a relatively small set for evaluation is highly unlikely when information sets are large. This process enables sampling a set of states that are deemed more likely through inference without slowing the player down catastrophically in large games.

Table 5.4 shows the effect of sampled subset size on tournament performance. Players ***500k** and ***100k** sample subsets with a maximum size of 500,000 and 100,000 states respectively. It turns out that sampling at most 100k states is sufficient for good performance in Skat with PIMC. Sampling more states does not consistently lead to significant improvement in tournament performance, and subset-sampling players perform comparably against Kermit as well. Experimenting with less states is unnecessary for PIMC players because the time spent estimating $\eta(s|I)$ with 100,000 states is negligible when compared to the PIMC state evaluations that follow. Table 5.5 shows that the time per action taken by the players that sample at most 100,000 states is similar to what was reported for Kermit in Table 5.2.

5.2.4 RecPIMC Cardplay Tournaments

Furtak and Buro [FB13] implemented a RecPIMC version of Kermit (RecKermit) and demonstrated that it was the state-of-the-art for computer Skat. Although it may be too computationally expensive for play against humans on commodity hardware, RecKermit was shown to be far stronger than regular

47

	Grand			Suit			Null		
	$n = 80$	$n = 160$	$n = 320$	$n = 80$	$n = 160$	$n = 320$	$n = 80$	$n = 160$	$n = 320$
RecDeep	39.19	38.99	38.45	20.81	20.95	21.96	13.41	13.21	11.76
RecKermit	38.93	39.2	38.65	19.16	18.70	18.55	10.69	10.37	9.37
Δ	0.26	-0.21	-0.20	1.65	2.25	3.41	2.72	2.84	2.39

Table 5.6: Tournament points/game for subset-sampling RecPIMC players separated by game type and number of states evaluated. RecDeep is the new state-of-the-art for Skat cardplay.

Kermit in the tournament setting. Furthermore, it was shown that RecKermit's workload is easily parallelizable — to the point where average move times can be reduced to a few seconds on 32 cores.

RecDeep is a modified version of RecKermit, in which Kermit's inference is replaced with individual card inference in both the top-level and lower-level players. The top-level player estimates $\eta(s|I)$ using a maximum of 100,000 states and the lower-level player estimates it using a maximum of 10,000.

Table 5.6 shows the average tournament points per game of each recursive player over 2500 matches for each game type and number of states evaluated. Top-level players evaluate n states and lower-level players evaluate 10. Standard deviations are 0.94, 0.78, and 0.76 in grand, suit, and null games respectively.

Although there is no significant difference in average tournament points per game observed for grands, RecDeep is significantly better in suit and null games. The lack of a difference in grand games can be explained by following the same logic as in Section 5.2.2. These results confirm that individual card inference has advanced the state-of-the-art in computer Skat cardplay.

Table 5.7: Average time (in seconds) taken per action by RecPIMC players on an Intel® Core™ i7-8700K processor.

Num States	80	160	320
RecKermit	77.56	157.02	254.82
RecDeep	112.69	211.00	388.19

A potential disadvantage of incorporating individual card inference into RecPIMC is that it increases the already expensive runtime cost of the algorithm — even when using a subset of states to estimate $\eta(s|I)$. Table 5.7 shows the time taken per action of each recursive player using a single core. The corresponding move times reported for RecKermit in Furtak and Buro [FB13] differ because they are computed using 32 cores. The cost of estimating $\eta(s|I)$ is substantially larger than in the PIMC case. Although the lower-level player estimates the distribution using only 10,000 states, it requires a forward pass of the network for each action taken by the lower-player. This amounts to $nm + 1$ forward passes to evaluate n top-level states with m actions remaining in the game as opposed to 1 in the PIMC case. However, this overhead could be reduced by batching forward passes from different lower-level players together. k threads simultaneously making lower-level player evaluations reduces the number of forward passes by a factor of k.

Chapter 6

Extensions to Individual Card Prediction

In this chapter, I propose and evaluate two possible extensions to the individual card location prediction algorithm discussed in Chapter 4. After empirical evaluation in the tournament setting, both proposed extensions lead to negative results, but their motivation is clear and the lessons learned provide a starting point for future work.

6.1 Accounting For Sequences

The models presented thus far do not use all available state information to make predictions. Although attempts at feature engineering seem to help, information related to the full action history is still lost during abstraction. In particular, important temporal information like when particular cards or tricks occur in relation to others is abstracted to reduce the network input size. A better representation of sequences could help capture this information, and may remove the need for domain knowledge in this approach altogether.

6.1.1 Flat Trick History Feature

The most basic approach to including the full action history is to simply encode it as a set of 32-card one-hot encoded actions concatenated together. There are 10 tricks, but only the first 9 form a decision point which means this leads to an additional $32 \times 26 = 832$ input features. With enough training data, it
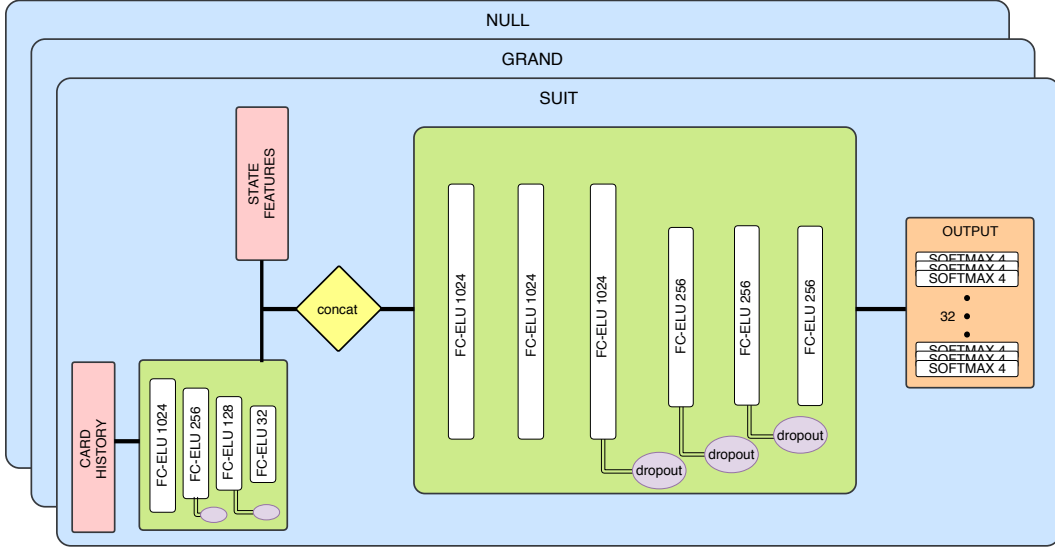
Figure 6.1: Inference network architecture with flat trick history module to capture sequential dependencies across cards and tricks.

may be possible to learn good representations from these raw inputs.

Due to its length, I provide the entire cardplay history (padded with zeros for future moves) as a separate input to the network. This helps reduce the total number of weights in the network and maintains a better balance between this feature and the rest of the input features. The input is fed through 4 separate hidden layers that reduce its dimensionality to 32, at which point it is concatenated with the rest of the state input features and fed through the rest of the network.

This approach significantly increases both the size of the input feature set and the number of weights required for the network, so it must result in large improvements for it to be considered worthwhile. It also has a significant impact on the physical size of states in the training set and massively increases training time by creating an I/O bottleneck.

After training, I observed no significant difference in *CE* (evaluated on 10^6 states) between using the flat trick history feature or not. Learning from raw input, especially about aspects that are difficult to capture using hand-engineered features, is an attractive prospect, but this basic technique proved unable to do so. Next, I show further empirical evidence confirming that this

	Grand			Suit			Null		
	$n = 80$	$n = 160$	$n = 320$	$n = 80$	$n = 160$	$n = 320$	$n = 80$	$n = 160$	$n = 320$
DeepHist	37.06	36.75	36.89	18.71	18.74	18.14	11.58	11.29	10.76
Deep	37.50	37.35	37.54	18.18	18.31	18.51	11.07	10.99	11.02
Δ	-0.44	-0.60	-0.65	0.53	0.43	-0.37	0.51	0.30	-0.26

Table 6.1: Effect of including flat trick history on tournament points/game.

feature does not improve inference quality by evaluating its performance in the tournament setting.

6.1.2 Tournament Results

DeepHist represents the proposed extension to Deep where the flat trick history feature is concatenated with the rest of the features from Table 4.2 This is the only difference between the two models. Average tournament points per game after 5,000 matches for each game type are shown in Table 6.1. One standard deviation amounts to 0.71, 0.56, and 0.53 tournament points per game for grand, suit, and null games respectively.

Due to differences between player tournament points being smaller than one standard deviation in all cases, it is clear that the additional cardplay history component did not improve performance much. This suggests that the module is not worth the additional training and space costs that it incurs. Forward passes of the network are also slower, but this has little impact on the final player's speed in this setting because only a single pass is made for any information set.

Sequential dependencies between cards and tricks could be better represented using a different network architecture. Specialized machinery for sequential learning like recurrent neural networks (RNNs) could potentially model dependencies that this approach does not. Although the engineered features are well-designed for this specific task, they are domain-specific and lead to predictions made on an abstraction of the information set rather than the actual information set. Replacing them with representations learned from raw input would make the overall approach to inference more general, and

```
Sinkhorn(Matrix L, List rowSums, List colSums, int k)
    for k iterations do
        L *= reduceSum(L, axis = 1)/colSums
        L *= reduceSum(L, axis = 0)/rowSums
    end
```

Algorithm 5: Generalized Sinkhorn-Knopp. `colSums` and `rowSums` are the desired sums for each column and row in matrix L.

could also lead to performance improvements by avoiding information loss through abstraction. This would be done by removing engineered state features related to which actions have been taken in the past and replacing them with representations learned from the raw sequences of actions in the game thus far.

6.2 Adding Structure to Network Output

In this framework, it is possible to enforce structure on network output to capture game-specific elements. For instance, some structure has already been enforced by using the softmax operator on the rows of L. This constrains the probability masses so that the sum for each card adds up to 1. In Skat, states are structured such that each player starts with 10 cards in their hand, and the remaining 2 cards are in the skat.

6.2.1 Sinkhorn-Knopp Algorithm

Sinkhorn's Theorem states that any $n \times n$ matrix with strictly positive entries can be scaled such that all rows and columns sum to 1. The matrix can be scaled by iteratively scaling columns and then rows until convergence. This is called the Sinkhorn-Knopp matrix scaling algorithm [Sin64; SK67], and is shown in Algorithm 5. Sinkhorn-Knopp has also been used as a distance metric for input features in machine learning [Cut13] and as a smooth optimal transport loss for generative models [GPC17].

The algorithm and its convergence guarantees generalize to any $m \times n$ matrix that is (r, c)-scalable. That is, if a matrix M can be scaled such that

$\sum_j M_{i,j} = r_i$ and $\sum_i M_{i,j} = c_i$ given $r \in \mathbb{R}^m$ and $c \in \mathbb{R}^n$, then it is scalable using Sinkhorn-Knopp. This generalization makes it possible to take a positive 32×4 matrix (the output from the inference network for Skat) and scale it so that all rows sum to 1 and the columns sum to 10, 10, 10, and 2 respectively. See [Ide16] and [CK18] for more details regarding convergence proofs for generalized Sinkhorn-Knopp.

By deciding on a fixed number of iterations, k, Sinkhorn-Knopp can be implemented as part of a deep neural network. All operations are differentiable, so libraries that provide automatic differentiation like Tensorflow make implementation simple. Hyperparameter k should be tuned to problem-specific requirements while considering that using more iterations slightly increases computational demands but can reduce the scaling error of the final matrix.

6.2.2 Training Results

To test whether applying Sinkhorn-Knopp helps enforce structure on the output distributions of a neural network and whether or not enforcing structure helps the final model, I implemented the same model presented in Section 4.2 with $k = 10$ Sinkhorn-Knopp iterations performed on the output distribution.

Table 6.2a shows how well state structure is implicitly captured by the original Logistic and Deep models, respectively. Deep appears to learn more about state structure implicitly in all game types.

LogisticSink and **DeepSink** represent the proposed extension to Logistic and Deep, and their rows in Table 6.2b shows the extension's effect on output structure. The expectations and standard deviations are computed by summing over the columns of L using a test set of 10^6 example states.

Implementing an iterative Sinkhorn operator leads to output distributions that appear to account for the structural information enforced by the algorithm. The standard deviations for the expected number of cards in the opponents' hands and the skat (when it is not known) are significantly lower — showing that output distributions have been successfully constrained to the desired structure by the Sinkhorn operator. However, this does not appear

	Grand				Suit				Null			
	Known		Unknown		Known		Unknown		Known		Unknown	
	Opp.	Skat	Opp.	Skat	Opp.	Skat	Opp.	Skat	Opp.	Skat	Opp.	Skat
Logistic												
μ	10.000	2.000	10.001	1.998	10.000	2.000	10.000	1.999	10.001	1.999	10.007	1.986
σ	0.587	0.009	0.617	0.395	0.470	0.004	0.546	0.440	0.660	0.038	0.648	0.389
Deep												
μ	10.000	2.000	9.970	2.059	9.999	2.001	9.983	2.033	9.997	2.006	10.005	1.991
σ	0.333	0.008	0.323	0.225	0.314	0.008	0.324	0.209	0.388	0.051	0.440	0.265

(a) Expected number of cards for original models.

	Grand				Suit				Null			
	Known		Unknown		Known		Unknown		Known		Unknown	
	Opp.	Skat	Opp.	Skat	Opp.	Skat	Opp.	Skat	Opp.	Skat	Opp.	Skat
LogisticSink												
μ	10.000	2.000	10.002	1.994	10.000	2.000	10.004	1.992	10.000	2.000	10.003	1.995
σ	0.068	0.009	0.074	0.031	0.067	0.003	0.086	0.041	0.067	0.016	0.078	0.039
DeepSink												
μ	9.999	2.001	9.998	2.003	10.000	2.001	9.998	2.004	10.000	2.001	10.002	1.995
σ	0.057	0.008	0.083	0.023	0.054	0.008	0.084	0.025	0.056	0.010	0.073	0.023

(b) Expected number of cards after $k = 10$ iterations of Sinkhorn-Knopp.

Table 6.2: Mean and standard deviation for the expected number of cards in opponent hands and skat for inference models.

to lead to any significant differences in *CE*, and it is unclear whether or not applying such constraints is beneficial to the model in practice. Next, I empirically test whether or not applying the Sinkhorn operator leads to better sampling distributions for determinized state evaluators in a practical tournament setting.

6.2.3 Tournament Results

Table 6.3 shows the effect of using the generalized Sinkhorn algorithm to structure network outputs on tournament points per game. One standard deviation amounts to 0.72, 0.56, and 0.53 tournament points per game in grand, suit, and null games respectively.

Score differences between Logistic and LogisticSink are smaller than one standard deviation. This suggests virtually no difference between the players.

	Grand			Suit			Null		
	$n = 80$	$n = 160$	$n = 320$	$n = 80$	$n = 160$	$n = 320$	$n = 80$	$n = 160$	$n = 320$
LogisticSink vs Logistic									
$P1$	38.27	38.30	38.15	20.73	20.55	20.56	12.82	12.74	12.71
$P2$	38.37	38.00	38.07	20.56	20.16	20.20	12.55	12.24	12.51
Δ	-0.10	0.30	0.08	0.17	0.39	0.36	0.27	0.50	0.20
LogisticSink vs Kermit									
$P1$	38.73	38.29	38.35	18.90	19.11	19.32	12.47	13.13	13.36
$P2$	38.03	38.21	37.97	21.42	20.97	20.85	12.93	12.10	12.11
Δ	0.70	0.08	0.38	-2.52	-1.86	-1.53	-0.46	1.03	1.25
DeepSink vs. Deep									
$P1$	37.73	37.37	37.03	19.21	18.11	18.72	11.57	11.36	10.96
$P2$	37.20	36.89	36.79	18.18	18.81	18.55	10.82	10.81	10.95
Δ	0.53	0.48	0.24	1.03	-0.70	0.17	0.75	0.55	0.01
DeepSink vs Kermit									
$P1$	38.89	38.59	38.50	20.87	20.25	20.60	14.11	14.3	14.19
$P2$	36.76	36.50	36.39	17.57	17.78	17.58	9.85	9.32	9.33
Δ	2.13	2.09	2.11	3.30	2.47	3.02	4.26	4.98	4.86

Table 6.3: Effect of structured output on tournament points/game.

Although DeepSink seems to outperform Deep when fewer states are sampled for evaluation, the extra structure provided by Sinkhorn does not appear to consistently improve tournament performance. Performance of the Sinkhorn models against Kermit is also roughly the same as their non-Sinkhorn counterparts. In this case, it seems that the structure learned implicitly from the training examples is enough, and that minimizing *CE* or *TSSR* may be more important than explicitly enforcing a specific structure on output distributions.

Chapter 7

Conclusion

In this thesis, I have shown that individual card inference trained using supervised learning can improve the performance of determinized algorithms in trick-taking card games considerably. This may not come as a surprise to seasoned Contract Bridge or Skat players as they routinely draw a lot of information regarding the whereabouts of remaining cards from past tricks. However, this thesis demonstrates how to do this using modern learning techniques. It shows how neural networks trained from human data can be used to predict fine-grained information like the locations of individual cards, and how to incorporate such predictions into current state-of-the-art search techniques for trick-taking card games — improving the current state-of-the-art Skat AI system significantly in the process.

This result is exciting and opens the door for future improvements. Playing strength could be increased by further improving inference so that the model can adjust to individual opponents and partners. State probability distributions could be smoothed to account for opponents who often make mistakes or those that play clever moves to confuse inference. A better technique for modelling sequential dependencies could substantially improve the techniques described in this work. Not only would it capture interactions between cards and when exactly they were played better than the engineered features presented in this work, but it could remove the need for domain knowledge in this task altogether.

Ultimately, determinized evaluation techniques are limited by their theo-

retical flaws and more theoretically-sound techniques should also be explored in future work. Although most of its guarantees no longer hold in 3-player or non-zero sum games, handcrafting or somehow learning a suitable abstraction so that a CFR-based technique can be applied is another promising but challenging avenue.

References

[Aba+16]   Martin Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. "Tensorflow: A system for large-scale machine learning." In: *OSDI*. Vol. 16. 2016, pp. 265–283.   31

[ACF02]   Peter Auer, Nicolo Cesa-Bianchi, and Paul Fischer. "Finite-time analysis of the multiarmed bandit problem." In: *Machine learning* 47.2-3 (2002), pp. 235–256.   21

[Bai+18]   Hendrik Baier, Adam Sattaur, Edward Powley, Sam Devlin, Jeff Rollason, and Peter Cowling. "Emulating human play in a leading mobile card game." In: *IEEE Transactions on Games* (2018).   23

[BS17]   Noam Brown and Tuomas Sandholm. "Superhuman AI for heads-up no-limit Poker: Libratus beats top professionals." In: *Science* (2017), eaao1733.   13

[Bro+12]   Cameron B Browne, Edward Powley, Daniel Whitehouse, Simon M Lucas, Peter I Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. "A survey of Monte Carlo tree search methods." In: *IEEE Transactions on Computational Intelligence and AI in games* 4.1 (2012), pp. 1–43.   1

[Bur97]   Michael Buro. "The Othello match of the year: Takeshi Murakami vs. Logistello." In: *ICGA Journal* 20.3 (1997), pp. 189–193.   2

[Bur+09]   Michael Buro, Jeffrey Richard Long, Timothy Furtak, and Nathan R Sturtevant. "Improving state evaluation, inference, and search in trick-based card games." In: *IJCAI*. 2009, pp. 1407–1413.   16, 20, 32, 41

[CHH02]   Murray Campbell, A Joseph Hoane Jr, and Feng-hsiung Hsu. "Deep Blue." In: *Artificial intelligence* 134.1-2 (2002), pp. 57–83.   2

[CK18]   Deeparnab Chakrabarty and Sanjeev Khanna. "Better and simpler error analysis of the Sinkhorn-Knopp algorithm for matrix scaling." In: *arXiv preprint arXiv:1801.02790* (2018).   54

[CUH15]   Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. "Fast and accurate deep network learning by exponential linear units (ELUs)." In: *arXiv preprint arXiv:1511.07289* (2015).   28

[CPW12]   Peter I Cowling, Edward J Powley, and Daniel Whitehouse. "Information set Monte Carlo tree search." In: *IEEE Transactions on Computational Intelligence and AI in Games* 4.2 (2012), pp. 120–143.   21, 22, 25

[Cut13]   Marco Cuturi. "Sinkhorn distances: Lightspeed computation of optimal transport." In: *Advances in neural information processing systems.* 2013, pp. 2292–2300.   53

[DOS18]   DOSKV. *DOSKV: Deutcher Online Skatverband.* https://www.doskv.de/skat-spielen.htm. 2018. URL: https://www.doskv.de/skat-spielen.htm.   31

[FB98]   Ian Frank and David Basin. "Search in games with incomplete information: A case study using Bridge card play." In: *Artificial Intelligence* 100.1-2 (1998), pp. 87–123.   17, 18

[FB13]   Timothy Furtak and Michael Buro. "Recursive Monte Carlo search for imperfect information games." In: *Computational Intelligence in Games (CIG), 2013 IEEE Conference on.* IEEE. 2013, pp. 1–8.   15, 20, 22–25, 47, 49

[GPC17]   Aude Genevay, Gabriel Peyré, and Marco Cuturi. "Learning generative models with sinkhorn divergences." In: *arXiv preprint arXiv:1706.00292* (2017).   53

[Gin01]   Matthew L Ginsberg. "GIB: Imperfect information in a computationally challenging game." In: *Journal of Artificial Intelligence Research* 14 (2001), pp. 303–358.   16, 20

[HI15]   Daniel Hennes and Dario Izzo. "Interplanetary trajectory planning with Monte Carlo tree search." In: *IJCAI.* 2015, pp. 769–775.   1

[Ide16]   Martin Idel. "A review of matrix scaling and Sinkhorn's normal form for matrices and positive maps." In: *arXiv preprint arXiv:1609.06349* (2016).   54

[KS06]   Levente Kocsis and Csaba Szepesvári. "Bandit based Monte-Carlo planning." In: *European conference on machine learning.* Springer. 2006, pp. 282–293.   21

[Lan+09]   Marc Lanctot, Kevin Waugh, Martin Zinkevich, and Michael Bowling. "Monte Carlo sampling for regret minimization in extensive games." In: *Advances in neural information processing systems.* 2009, pp. 1078–1086.   14

[LBH15]   Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. "Deep learning." In: *nature* 521.7553 (2015), p. 436.   20, 26

[Lev89]   David NL Levy. "The million pound Bridge program." In: *Heuristic Programming in Artificial Intelligence* (1989).   16

60

[Lin18]   Ron Link. *Edmonton Skat Club.* https : / / skatlink . com /
           contact_us.html. 2018. URL: https://skatlink.com/contact_
           us.html.   41

[Lon+10]   Jeffrey Richard Long, Nathan R Sturtevant, Michael Buro, and
           Timothy Furtak. "Understanding the success of Perfect Informa-
           tion Monte Carlo Sampling in game tree search." In: *AAAI.* 2010.
               3, 18, 19

[Mor+17]   Matej Moravčík, Martin Schmid, Neil Burch, Viliam Lisỳ, Dustin
           Morrill, Nolan Bard, Trevor Davis, Kevin Waugh, Michael Johan-
           son, and Michael Bowling. "Deepstack: Expert-level artificial in-
           telligence in heads-up no-limit Poker." In: *Science* 356.6337 (2017),
           pp. 508–513.   13, 14

[Nas50]   John F Nash. "Equilibrium points in n-person games." In: *Pro-
           ceedings of the national academy of sciences* 36.1 (1950), pp. 48–
           49.   12

[Ope18]   OpenAI. *OpenAI Five.* https : //blog . openai . com/openai-
           five/. 2018.   1

[Pre98]   Lutz Prechelt. "Automatic early stopping using cross validation:
           quantifying the criteria." In: *Neural Networks* 11.4 (1998), pp. 761–
           767.   28

[RN16]   Stuart J Russell and Peter Norvig. *Artificial intelligence: a modern
           approach.* Malaysia; Pearson Education Limited, 2016.   17

[Sch+07]   Jonathan Schaeffer, Neil Burch, Yngvi Björnsson, Akihiro Kishi-
           moto, Martin Müller, Robert Lake, Paul Lu, and Steve Sutphen.
           "Checkers is solved." In: *science* 317.5844 (2007), pp. 1518–1522.   2

[SBH08]   Jan Schafer, Michael Buro, and Knut Hartmann. "The UCT
           algorithm applied to games with imperfect information." In:
           *Diploma, Otto-Von-Guericke Univ. Magdeburg, Magdeburg, Ger-
           many* (2008).   22

[Sch+17]   John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford,
           and Oleg Klimov. "Proximal policy optimization algorithms." In:
           arXiv preprint arXiv:1707.06347 (2017).   1

[Sil+16]   David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Lau-
           rent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis
           Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. "Master-
           ing the game of Go with deep neural networks and tree search."
           In: *nature* 529.7587 (2016), p. 484.   2, 20

[Sil+17a] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. "Mastering Chess and Shogi by self-play with a general reinforcement learning algorithm." In: *arXiv preprint arXiv:1712.01815* (2017).   20

[Sil+17b] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. "Mastering the game of Go without human knowledge." In: *Nature* 550.7676 (2017), p. 354.   2, 20

[Sin64] Richard Sinkhorn. "A relationship between arbitrary positive matrices and doubly stochastic matrices." In: *The annals of mathematical statistics* 35.2 (1964), pp. 876–879.   53

[SK67] Richard Sinkhorn and Paul Knopp. "Concerning nonnegative matrices and doubly stochastic matrices." In: *Pacific Journal of Mathematics* 21.2 (1967), pp. 343–348.   53

[Sri+14] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. "Dropout: A simple way to prevent neural networks from overfitting." In: *The Journal of Machine Learning Research* 15.1 (2014), pp. 1929–1958.   28

[Stu08] Nathan Sturtevant. "An analysis of UCT in multi-player games." In: *ICGA Journal* 31.4 (2008), pp. 195–208.   16, 22

[SW06] Nathan R Sturtevant and Adam M White. "Feature construction for reinforcement learning in Hearts." In: *International Conference on Computers and Games*. Springer. 2006, pp. 122–134.   16

[SGS13] Duane Szafron, Richard Gibson, and Nathan Sturtevant. "A parameterized family of equilibrium profiles for three-player Kuhn Poker." In: *Proceedings of the 2013 international conference on Autonomous agents and multi-agent systems*. International Foundation for Autonomous Agents and Multiagent Systems. 2013, pp. 247–254.   14

[Tes95] Gerald Tesauro. "Temporal difference learning and TD-Gammon." In: *Communications of the ACM* 38.3 (1995), pp. 58–68.   2

[Wau+15] Kevin Waugh, Dustin Morrill, James Andrew Bagnell, and Michael Bowling. "Solving games with functional regret estimation." In: *AAAI*. Vol. 15. 2015, pp. 2138–2144.   14

[WKW70] Frank Wilcoxon, SK Katti, and Roberta A Wilcox. "Critical values and probability levels for the Wilcoxon rank sum test and the Wilcoxon signed rank test." In: *Selected tables in mathematical statistics* 1 (1970), pp. 171–259.   41

[Zin+08]   Martin Zinkevich, Michael Johanson, Michael Bowling, and
             Carmelo Piccione. "Regret minimization in games with incom-
             plete information." In: *Advances in neural information processing
             systems.* 2008, pp. 1729–1736.   2, 12