University of Alberta

Search, Inference and Opponent Modelling in an Expert-Caliber Skat Player

by

Jeffrey Richard Long

A thesis submitted to the Faculty of Graduate Studies and Research
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

Department of Computing Science

Fall 2011
Edmonton, Alberta

Search, Inference and Opponent Modelling in an Expert-Caliber Skat Player

Jeffrey Richard Long

Abstract

In this dissertation, we investigate the problems of search, inference and opponent modelling in imperfect information games through the creation of a computer player for the popular german card game skat. In so doing, we demonstrate three major contributions to the field of artificial intelligence research in games. First, we present our skat player Kermit which, using a synthesis of different techniques, decisively defeats previously existing computer players and displays playing strength comparable to human experts. Second, we propose a framework for evaluating game-playing algorithms with known theoretical flaws and explaining the success of such methods in different classes of games. Finally, we enhance Kermit with a simple but effective opponent modelling component that allows it to adapt and improve its performance against players of differing playing strength in real time.

Acknowledgements

First and foremost, I would like to acknowledge my supervisor Dr. Michael Buro for his guidance, support, and innumerable lessons about how to lose to him at skat. Dr. Nathan Sturtevant is also deserving of my thanks for valuable discussion and advice, as is my fellow student Tim Furtak. I also express thanks to Nathan Taylor and Ryan Lagerquist for their general work on the skat project at various points.

I thank also my family for their boundless support and belief in the value of education.

Finally, I would like to thank Shavaun Liss, Ben Willson and many other friends who have indulged me in my fanatical devotion to game-playing at any point — and even for occasionally letting me win.

Table of Contents

The beginning is the most important part of the work.

-Plato

1 Introduction

The study of competitive games has been one of the most enduring and challenging problems in the field of artificial intelligence. In the early 1950's, pioneering computer scientists such as Turing and Shannon [39] were already studying the game of chess, and Samuel's introductory work on checkers began later that same decade [36]. These lines of study eventually culminated in the Deep Blue super computer's famous defeat of chess Grand Master Kasparov, and in the creation of the program Chinook which claimed the world championship title in Checkers [37]. A defining characteristic of checkers, chess and nearly all other games studied for the 40 years following Turing and Shannon's work is that they are games of *perfect information*. A perfect information game is one in which all players have complete knowledge of the state of the game at all times. By contrast, the study of games of *imperfect information* — games in which at least one player does not have full knowledge of the game state — has lagged behind the study of perfect information games by nearly 50 years. Only in the late 1990's did serious work emerge for the game of bridge [15] [41] [9], and poker followed as a popular domain of study shortly thereafter [2]. While there has been significant progress in the past 10 years, the arena of imperfect information games has yet to see the kind of decisive defeats of human world champions that have occurred in the perfect information domain.

Research in games is often motivated by the claim that games make an ideal testing ground for more general artificial intelligence techniques. Indeed, games have some attractive features for this purpose: they have well-defined rules, a finite state and action space, and offer a straightforward way of measuring performance. This latter feature is particularly important in the context of imperfect information. Consider real-world adversarial settings that might conceivably be modelled as an imperfect information game: election strategy, business and investment strategy, or military planning and counter-insurgency. Often, such environments have uncertainties that will never be resolved even after the fact, making performance evaluation of any policy a difficult task, and trial-and-error is typically costly if not impossible. It is therefore all the more useful to be able evaluate techniques for dealing with these environments in the well-behaved model that games provide.

However it is the belief of this author that games are not solely useful as incubators for ideas that will some day grow up to be useful in the real world. Games *are* a real-world domain themselves. The commercial success of the video game industry is often used as evidence of this point, but of course games have played a significant cultural role for centuries before the first computer was invented. In many ways, games are the mental analog of physical sport. In addition to provid-

ing mental exercise and recreation, traditional face-to-face games are often lauded for their social aspects. However, these same aspects can make them daunting to beginners. Rather than embarass themselves in front of their peers, many beginning players prefer to learn games playing against a computer who does not mind if the novice spends some extra time taking their turn (human players, on the other hand, are rarely so patient). What's more, the success of computer players in games is an achievement that even people lacking a scientific or technical background can understand and with which they can connect and identify. We therefore suggest that games are valuable to artificial intelligence research both as test-tube and as show-case.

Imperfect information is one important feature that separates games that we know, at least in general terms, how to approach from those that we do not. Moreover, as imperfect information is a near-ubiquitous property of real-world scenarios, it is a feature that must be well understood in order to lift artificial intelligence techniques used in games to other applications. Imperfect information poses some distinct challenges for decision making agents. One such challenge is how to evaluate the merits of different actions when the true state of the environment is partially unknown. Another is determining information that is implicitly conveyed by the actions of other agents in the environment, and also reasoning about the information conveyed by our own actions. We take some steps to address both of these challenges over the course of this work.

1.1 Problem Domain

In this dissertation, we study the problem of imperfect information in games through the creation of an expert-caliber computer player for the game of skat. Skat is a three-player, trick-taking card game that is similar to bridge, its better known British and North American counterpart. In skat, players first compete in an auction for the priviledge of becoming the "soloist" while the remaining two players become joint defenders, and then play out their cards to determine whether or not the soloist wins her game. We describe skat in much more detail in Chapter 2. For now, it suffices to say that skat is a challenging domain that exemplifies many of the key aspects of imperfect information games. Prior to our work here, computer systems existed that could play passably well during the cardplay portion of skat, but bidding during the auction phase was a major weakness, and no existing computer skat player could claim comparable playing strength to the level of human experts in terms of playing the entire game.

1.2 Contributions

Our work described here makes the following three major contributions to the study of imperfect information games. Each of these contributions has also been published in an associated paper.

- **The creation of an expert-level computer skat player.** We construct a computer player for the game of skat that decisively defeats previous computer players and plays at approximately

the same level as expert humans. A combination of three core aspects — game-tree search, state evaluation and inference of hidden information — are used to achieve this performance. We describe the different techniques used to create each one. This work has been published by Buro et al. [4].

- **A framework for understanding suboptimal solution methods.** A key contributor to our program's strength is an algorithm known as Perfect Information Monte Carlo (PIMC) search. Although it has often produced strong results in imperfect information games, PIMC search is a suboptimal solution technique with known theoretical flaws. We present a framework for understanding and analyzing the success of PIMC search, and for predicting classes of games where such a method is likely to be successful. This work has been published by Long et al. [26].

- **A simple technique for adaptive opponent modelling in real time.** Computing a strong, static solution to a complex imperfect information game is challenging, but creating a dynamic player that adapts to exploit its opponents' weaknesses in real time is perhaps harder still. We present an enhancement to our skat player that allows it to adapt to individual opponents. Although our player only adapts one simple aspect of its behaviour, it is able to do so in real time after only a very small number of encounters with an opponent. This adaptation results in substantial gains against some opponents without compromising playing strength in self-play. This work has been published by Long and Buro [27].

The remainder of this document is organized as follows. Chapter 2 presents a brief review of game theoretic terms and notation, and a detailed overview of the game of skat. Chapter 3 reviews previous work and existing techniques for playing imperfect information games. Chapter 4 describes the first of our main contributions, the construction of our expert-level computer player Kermit. Chapter 5 describes our framework for analyzing the success of PIMC search. Chapter 6 is devoted to discussion and analysis of the problem of inference as it relates to imperfect information games — the content of this chapter has not previously appeared outside of this dissertation. Chapter 7 presents our real-time opponent modelling technique. Chapter 8 gives an overview of an assortment of techniques that we investigated but that failed to produce significant successful results. These ideas may be of interest for the purpose of future work, or perhaps just as cautionary tales. Like Chapter 6, this content has also not been published. Finally, Chapter 9 concludes the dissertation.

Oh the places you'll go! There is fun to be done! There are points to be scored. There are games to be won.

-Theodor Geisel, alias Dr. Seuss

2 Background

In this chapter, we first review basic game theoretic concepts and terminology that will be used throughout this document. We then outline the rules of the game of skat, our primary application domain, and then provide an analysis of why skat is an interesting game to study.

2.1 Game Theoretic Concepts

2.1.1 Extensive-Form Games

An *extensive-form game* is a formal, general model of a multiagent decision-making environment. It is computationally convenient for modelling environments where agents take actions in turn, as the action sequence is represented explicitly. An extensive-form game G consists of the following components:

- A finite set P of N rational players.
- A tree T consisting of a set of states S and a set of actions A that define transitions between states. This is often called the *game tree*.
- A set of terminal states, Z.
- Each terminal state $s \in Z$ has an associated N-tuple of payoffs, denoting the utility that each player $p \in P$ will receive in that state. We expect that the goal of each player is to maximize her own utility in the game.
- A partition of the game's non-terminal states into $N + 1$ subsets (some of which may be empty), indicating the player to act in each non-terminal state. Player $N + 1$ is a special player called *Chance* whose strategy is fixed and known to all players.
- A further partitioning of the states belonging to each player p_i into *information sets*, $\mathcal{I}_i$. Let $I_i \in \mathcal{I}_i$ be an information set belonging to player $p_i \in P$. Then for any two states $s \in I_i$ and $s' \in I_i$, the player p_i is unable to distinguish s from s' and must therefore act identically in both states.

Within this formal model, player p_i selects her actions according to her *strategy*, σ_i. σ_i is a mapping from each information set I_i to a probability distribution over all the actions available in

I_i. We call σ_i a *pure strategy* if the probability it assigns to playing any action in any information set is either 0 or 1. In other words, if player i is playing a pure strategy, then given the moves of the other players (including chance), player i's actions are entirely deterministic. We call σ_i a *mixed strategy* if for at least one action a for some information set I_i, σ_i assigns to a a probability in the interval $(0, 1)$. In such instances, player i is assumed to throw a random number "in her head" to select her action and thus will sometimes make different moves in identical situations.

It is worth noting that certain types of extensive form games have commonly used special names based on their properties. We list some of these below.

- A *perfect information game* is a game that can be represented in such a way that the number of game tree states in every information set is exactly one. An *imperfect information game* contains at least one information set of size greater than one.
- A *deterministic game* is one in which the special player Chance plays no actions, although we note that the outcome of such games can be non-deterministic since players are permitted to play mixed strategies. A *stochastic game* is one in which the Chance player plays at least one action somewhere in the game tree.
- A *multi-player game* is commonly used to refer to a game with *more than* two players.
- A *finite game* is one in which the game is guaranteed to reach a terminal state in a finite number of moves.
- A *zero-sum* game is a game in which at every terminal state, the sum of the payoffs for all players is zero. The term *constant-sum game* is used to describe a game where the payoffs for all players always sums to the same constant; however since payoffs can always be normalized, a constant-sum game can be equivalently represented as a zero-sum game. Zero-sum game will thus be the term of choice used in this document.

2.1.2 Solution Concepts

In the setting of a formal game, we are often interested in the question of "how should a player act so as to maximize her expected payoff?" The most common game theoretic answer to this question is that players should aim to play a *Nash equilibrium*. A strategy profile σ (specifying a strategy σ_i for each player i) is said to be in equilibrium if no player can improve her expected payoff by *unilaterally* changing her individual strategy, σ_i.

As proven by Nash, at least one (and often, many more than one) Nash equilibrium exists for every finite game with perfect recall so long as mixed strategies are allowed [30]. In the special case of two-player, zero-sum games, equilibrium strategies have some important properties that make them especially attractive as a solution concept. In a two-player, zero-sum game, if the strategy pair (σ_1, σ_2) is an equilibrium, and the strategy pair (σ_1', σ_2') is an equilibrium, then (σ_1, σ_2') and (σ_1', σ_2)

are also in equilibrium. In other words, in this special setting, equilibrium strategies for individual players are fully interchangeable and may be mixed and matched without changing the expected outcome of the game for either player. This property does not necessarily hold in non-zero-sum games or games with more than two players.

Numerous methods for finding Nash equilibria exist, the Lemke-Howson algorithm being among the most common [24]. Such methods traditionally require that the game being solved be represented in its *normal form*, an N-dimensional matrix where each matrix entry represents the terminal payoffs that are obtained by each player playing a single pure strategy. The size of this matrix is exponential in the size of the game tree (which is itself usually very large). In 1994, Koller et al. presented a solution technique that allows for a linear, rather than exponential, representation of the game tree [22], and in 2008, Zinkevich et al. proposed Counterfactual Regret minimization (CFR), which further reduces the memory requirements of finding equilibria to be linear in the number of information sets, as opposed to game states [50]. Due to these advances, the size of games that can be efficiently solved has increased drastically in recent years; current techniques can solve games of approximately 10^{12} game states, although these are typically limited to two-player, zero-sum games. We will discuss these and other techniques further in Chapter 3.

Finally, we should note that playing (or attempting to play) an equilibrium strategy may not always be desirable, even in zero-sum two-player games. While an equilibrium guarantees the best possible "worst-case" payoff against strong opposition, it does not necessarily benefit from mistakes by the opponent. In fact, depending on the structure of the game, an equilibrium strategy may be so "defensive" that it becomes difficult or even impossible for an opponent to make a mistake in the first place.

2.2 Skat Rules of Play

In Germany and its surrounding regions, skat is the card game of choice, boasting over 30 million casual players and more than 40,000 players registered in the International Skat Player's Association (ISPA) `http://www.ispaworld.org/`. Although much less prevalent than contract bridge in the English-speaking world, there are nevertheless skat clubs in most major Canadian cities.

Skat is a trick-taking card game for three players. It uses a "short" 32-card playing deck, similar to the standard 52-card playing deck except all cards with numeric value two through six have been removed from each of the four suits. The game is played over a series of independent hands; typically, hands are grouped into sets of 36 which are known as lists. Each player's objective is to maximize the number of points she makes over all her hands. A hand begins with each of the three players being dealt ten cards. The remaining two cards are called the skat, and are set aside for later without being examined by any player. The ordering of the players is also important; the player to the dealer's left is called the "forehand" player, with the remaining players being called "middlehand" and "rearhand" respectively (rearhand is also normally the dealer as well).

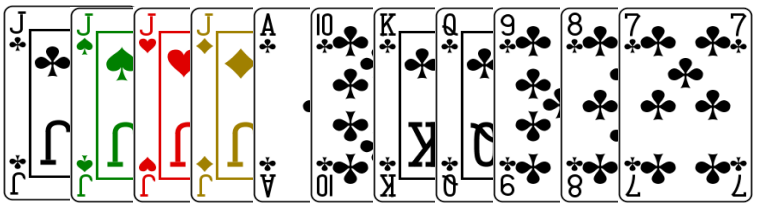


Figure 2.1: Ranking of cards in the trump suit.

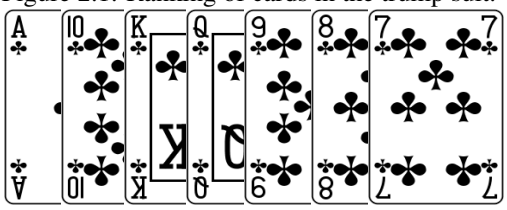


Figure 2.2: Ranking of cards in non-trump suits.

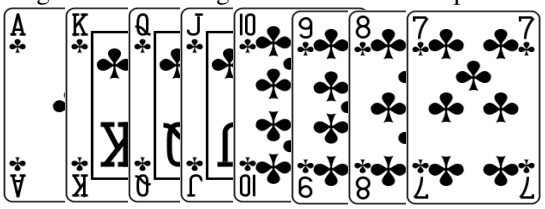


Figure 2.3: 'Standard' card ordering, used in null games.

Once the players have been dealt their cards, the remainder of the hand consists of two phases. The first is the bidding phase, in which players compete to be the "soloist" of the hand, a position analogous to the declarer in bridge. Unlike bridge, there are no permanent alliances of players between hands, but the two players who lose the bidding will become partners for the remainder of the current hand. The relative ranking of cards also differs from bridge; in particular, the four Jacks are generally special cards and make up the highest-ranked *trump* cards in the game, and are usually a member of the trump suit, rather than the suit depicted on the card. In addition, the rank of the Jacks relative to each other is fixed, with the clubs Jack being the highest rank, followed by spades, hearts and diamonds in that order. Within suits, the ranking of cards is slightly altered from the standard ordering, with the Ten ranking just below the Ace and ahead of the King. These rankings are depicted in Figure 2.1 for the trump suit and Figure 2.2 for non-trump suits. In certain special instances, however, the rankings of the cards reverts to "standard" bridge ordering, which we depict in Figure 2.3. We will discuss how to determine which ranking to use after describing the bidding process.

Players make bids in terms of a numeric *game value*. If a player places a bid of B, it is akin to saying "If I win the bid, I will announce a game of value at least B". Calculating the value of the game relies on two factors: a *base value* that depends on the game type that the soloist eventually announces, and a *multiplier*, which depends largely on which Jacks the soloist is holding. The final

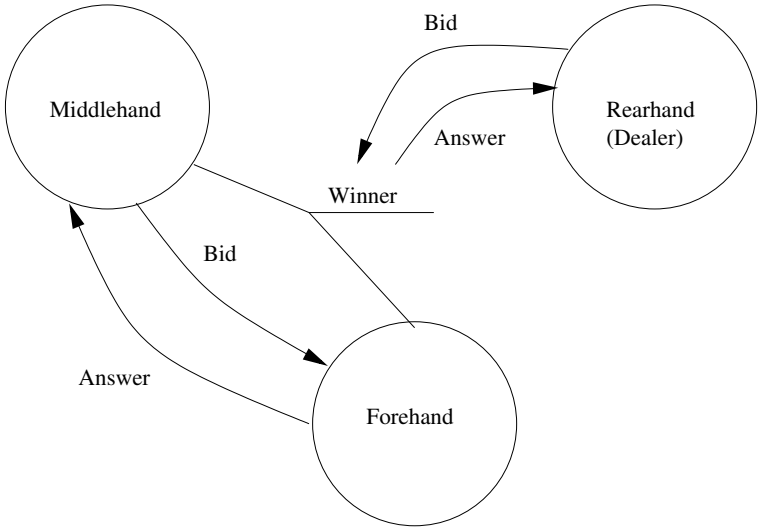


Figure 2.4: The bidding sequence in skat. First middlehand bids with forehand, then rearhand continues the bidding with the winner of the two.

game value is obtained by simply multiplying these two factors together. Players place bids in a fixed order depending on their position. Middlehand starts the bidding by announcing a bid B. Forehand then responds with either "yes", indicating she will also play a game of at least value B, or "pass." If forehand said "yes", then middlehand either announces a higher bid, or passes. In either case, when one player has passed, rearhand continues the bidding by assuming the role of announcing the bids, while whichever of forehand or middlehand remains answers either "yes" or "pass" to each one. Whichever player does not pass during this process becomes the soloist, or declarer, while the other two players form a temporary alliance and are now called the defenders. This bidding process is graphically depicted in Figure 2.4.

Once the bidding is over, the soloist has the option to pick up the skat — the two cards which were set aside at the beginning of the hand. If she does so, she may take the two cards into her hand, and then discard any two cards that she likes (including one or both of the cards she just picked up). These discarded cards are considered to have been 'won' by the soloist for purposes of determining who will win the hand. If the soloist opts not to pick up the skat, she is playing a *hand* game, which increases the multiplier to her game value by one.

After either picking up the skat and discarding cards or opting to play a hand game, the soloist then announces the *game type*. There is one game type for each of the four suits (diamonds, hearts, spades and clubs), with base values of 9, 10, 11 and 12 respectively. These together are referred to as *suit games*. Other game types include *grand*, with a base value of 24, and *null*, with a base value of 23. Null games are special, however, in that they have no multiplier to their base value. A null-hand game, where the soloist opted not to pick up the skat, is worth 35, and null-ouvert, where the soloist lays his cards on the table for the defenders to see, is worth 46. Finally, the soloist may

Table 2.1: Base game values for different skat game types.

Game Type	Base Value
♢	9
♡	10
♠	11
♣	12
Null	23
Grand	24
Null-hand	35
Null-ouvert	46
Null-ouvert-hand	59

play null games both ouvert and hand, which has a value of 59. These game values are summarized in Table 2.1.

In any game except null games, the four Jacks are considered the highest trump cards and are a member of the trump suit. In grand games, the Jacks are the only trumps; in suit games, all cards of the named suit are trump as well. In null games, however, there are no trumps and the card ranks revert to their "standard" ordering. In non-null games, the trump cards held by the soloist determine her game value multiplier. If the soloist is holding the Jack of clubs, she counts the number of consecutive trumps she holds in descending order of rank and adds 1 to the total to get her final game value multiplier. An example of this is shown in Figure 2.5. If she is not holding the Jack of clubs, she counts how many consecutive trumps she is *missing* from the top, and again adds one to obtain her game value multiplier. An example of such a hand is shown in Figure 2.6.

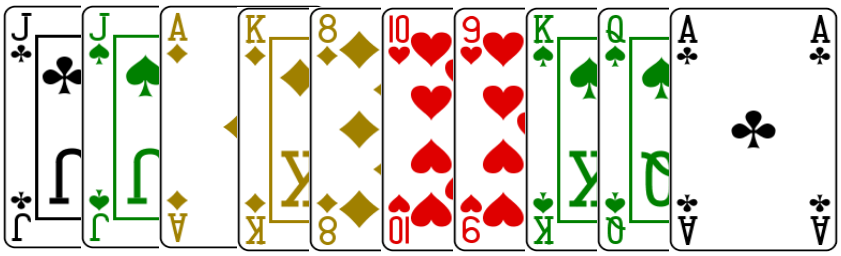


Figure 2.5: This hand contains the clubs and spades Jacks, but is missing the hearts Jack. Therefore, we say this hand is "with two" for a final game multiplier of three.

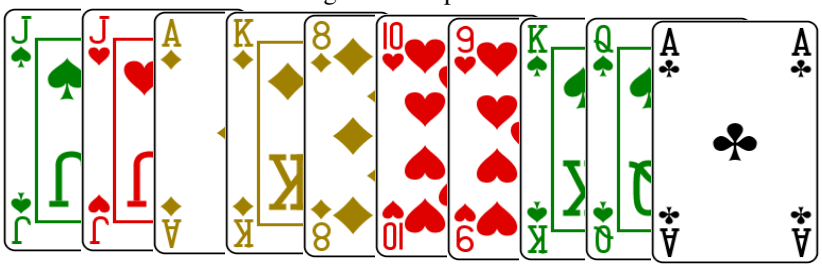


Figure 2.6: This hand is missing the clubs Jack, but contains the spades Jack. Therefore, we say this hand is "without one" for a final game multiplier of two.

Once the soloist has chosen the game type, the cardplay phase of the game begins. This phase takes place over a series of 10 *tricks*, with each player playing one card per trick. The player leading the trick may play any card in her hand; the other players must then follow suit if they are able to do so. If unable to follow suit, the other players may play any card. If one or more trump cards are played, the trump with the highest rank takes the trick; otherwise, the card with the highest rank in the suit that was originally lead takes the trick. The player seated in forehand always leads the first trick, regardless of who won the bidding, and afterwards each trick is lead by the winner of the previous trick.

The objective of the soloist during cardplay is to take 61 out of 120 available *card points*. These points are completely distinct from the *value* of the game that was used for bidding. Each card is worth a fixed number of card points: Aces are worth 11, Tens are worth 10, Kings are 4, Queens are 3 and Jacks are 2. All other cards are of no value. If the soloist succeeds in taking 90 card points, she is said to have played the opponents "schneider" and may add one to her game value multiplier. Again, null games are an exception, where card points are irrelevant and the soloist's objective is to *lose* every trick. If the soloist fulfills these objectives, then at the end of the hand she receives points equal to the value of the game she ended up playing, so long as this value is greater than or equal to her bid in the auction phase. If she fails, then she loses *double* the value of the game that was played. In most tournament and club settings, soloist wins are worth an additional constant bonus of 50 points, and soloist losses are given an additional 50 point penalty. Finally, if the soloist loses the hand, then the defenders score a small, constant number of game points — 40 or 30, depending on whether 3 or 4 players sit at the table respectively (in the case of a 4-table, the dealer also earns these defender points for ease of physical book-keeping).

2.3 Properties of Skat

Skat is an interesting game to study for several reasons beyond its popularity. First of all, the sheer size of skat's game tree is immense. There are 2,753,294,408,504,640 opening deals. Each such deal could result in one of 6 different major game types, for which any of the 3 players could take the role of the soloist. The number of ways in which each deal can be played differs by deal and game type, but is at least $10!$ in all cases, since the player to lead can always play any card from her hand. This gives us an extremely conservative lower bound of $1.7 \cdot 10^{23}$ states, and we have not even accounted for the auction phase, or certain special-case game declarations. This puts skat far outside the range of games that can be solved in a game theoretic sense.

Another crucial property of skat, from a complexity standpoint, is that it is in fact a multi-player game. For cardplay, since the players are partitioned into only two teams, the game can somewhat safely be abstracted as a two-player game. This is, at least, typically the approach that has previously been taken in the game of bridge (although results by von Stengel and Koller [47] show that such an abstraction poses pitfalls even in this relatively simple case, which we discuss further in Chapter 3).

However, unlike in bridge, the bidding phase is a three-way competition, which gives us a venue to explore some of the challenges posed by multi-player games.

For the cardplay phase, other authors have treated skat as a two-player, zero-sum game [38] [23]; as we have noted in our game theory overview, the zero-sum property is a particularly important one. In most of our work here, we will make that same assumption. However, we must be aware that, strictly speaking, this assumption is *not* true. This is due to a peculiar artifact in the way that skat games are scored, namely in that the game value won by the soloist is partially determined by her cards (in particular, how many of the highest trump cards she is playing with or without). Recall that defenders score a small, constant number of points for defeating the soloist. Consider now a situation late in the game where a defending player has two possible plays: one which wins the game if the soloist holds the ♣J and the other wins if his partner holds the ♣J instead. Let us say that the game value is much higher if the soloist holds the ♣J, but based on the bidding and other information, the defender believes it is slightly more likely that the soloist does not hold the Jack. We now have a situation where our defending player must choose between maximizing points for himself — making the play that is slightly more likely to earn his defender points — and minimizing the score of the soloist. In practice this dilemma appears to be rare, but it is yet another factor that makes skat a difficult game to deal with.

The bidding phase poses other challenges, in the sense that it appears to be the sort of problem to which standard search techniques are poorly suited. One reason for this is that while existing search methods are reasonably effective at choosing good moves during cardplay, they do not always accurately predict the end result of the game. Furthermore, existing techniques usually fail to provide a meaningful winning probability associated with moves, which is particularly relevant for bidding. We will discuss this claim further in future chapters. Bidding decisions may also be affected by inferences a player might make concerning the contents of the skat, based on the bidding of the other players. Finally, unlike in bridge where it is required to be able to explain the "meaning" of bids in plain English, in skat a player has complete freedom to bid as she wishes. This is convenient for quickly creating computer agents that can compete against humans, as well as offering more flexibility in creating a machine-optimized bidding process.

Skat possesses one more wrinkle from an inference and cardplay perspective. In many imperfect information games, such as bridge, poker and blackjack, the only moves which are not observed by all players are those made at chance nodes — for example, the dealing out of individual hands of cards. Skat, however, also contains hidden moves made by actual players — namely, the soloist's discard of two cards after picking up the skat. Hidden information resulting from chance nodes seems, intuitively at least, relatively easy to deal with, since we can sample from the known distributions of those chance events. With hidden information resulting from opponent moves, we have no such convenient starting point, which has the potential to raise interesting challenges.

A good leader needs all *the information.*

-My father, CONSTANTLY

3

Previous Work in Imperfect Information Games

In this chapter, we review previous work in the domain of computer players for imperfect information games. As work on skat in particular has been fairly limited, we include work on a variety of different imperfect information domains. Although the list we present here is by no means exhaustive, it represents a reasonable snap-shot of current techniques.

3.1 Contract Bridge

The most extensive body of work concerning an imperfect information game closely related to skat is on the game of contract bridge. Bridge is similar to skat in many ways; it is a trick-taking card game consisting of a bidding phase followed by a cardplay phase; and it is played in teams such that players do not know the hands of either their opponent(s) or their partner. We describe here some of the historical work in this domain, as well as the basic techniques that will make up the foundations of our computer skat player.

3.1.1 Planning Techniques for Bridge

One early success in computer bridge playing arose from the use of AI planning techniques by Smith et al. [41], who employed Hierarchical Task Networks (HTNs) to construct a computer player for declarer cardplay. Instead of searching over the enormous space of possible cardplay paths, their program, Tignum 2, plans over an abstract space of tactical ploys, many of which are commonly described and used by human experts. These techniques were incorporated into a commercial program called the Bridge Baron, which won the 1997 Baron Barclay World Bridge Computer challenge. The HTN planning approach was not used, however, for bidding or defender cardplay, and Smith et al. report that as of their 1997 victory, the overall performance of computer bridge programs was still far shy of world-class human play.

3.1.2 GIB: A Player for Contract Bridge

In 1998, Ginsberg proposed an alternative approach to computer bridge with his player called GIB [15]. Rather than rely on planning and attempts to mimic human play, GIB employed a brute force approach in the form of *Perfect Information Monte Carlo* (PIMC) search, a technique first suggested by Levy [25]. Conceptually, the algorithm is very simple. Whenever the PIMC player is to move,

we first create a hypothetical world, h, where we assign a value to all unobserved variables in the current game state. In a card game like bridge, this would consist of distributing cards between the unobserved hands of our opponents. Once this information has been assigned, we then assume that all players have perfect information of the game and employ standard perfect information search techniques to determine the best move at the root for that world h. We record the utility of the move according to the search and then repeat this process several times, choosing a different h for each iteration. In the end, we simply play the move at the root which has the highest average utility. Using this technique, by 2001, GIB was claimed to be the strongest computer bridge player in the world and of roughly equivalent playing strength to human experts (at least for its cardplay).

The linchpin of GIB's success is its ability to solve an entire 52-card deal of the perfect information (often called *double dummy*) variant of the game of bridge. Minimax search with alpha-beta pruning alone is too slow to solve such deals, or at least to solve a sufficient quantity of them for the Monte Carlo sampling procedure. To overcome this speed deficiency, Ginsberg introduces the concept of *partition search*. Partition search is an extension of the well-known game search technique of transposition tables. Transposition tables store individual game positions that can potentially be reached by numerous paths through the game tree, and a game outcome associated with the stored positions. Ginsberg's partition search extends transposition tables to store *sets* of positions associated with a game outcome, instead of individual positions. This technique requires the creation of certain game-specific functions and abstractions (for example, treating small cards as "don't cares" in bridge), and allows GIB to solve full 52-card deals several orders of magnitude faster than was previously possible.

In light of the fact that bridge is not a perfect information game, Ginsberg improves GIB's cardplay further by introducing the concept of alpha-beta search over lattices. This allows the program to search over *sets* of card configurations in which the declarer makes the contract, instead of over the numeric interval normally used in evaluation functions. This allows GIB to capture the imperfect information of bridge to a much greater extent; however, Ginsberg only uses this technique for GIB's declarer card play, and it still assumes that the defenders have perfect information of the game.

The majority of Ginsberg's work focuses on the search techniques used for GIB's cardplay. Much less is said about the creation of hypothetical worlds used by the Monte Carlo simulations, or the bidding process that occurs at the start of every bridge hand. For bidding, GIB uses an enormous database of hand-crafted rules to suggest candidate bids in a given situation, followed by *Borel simulations* in order to distinguish between 'close' decisions. Borel simulations construct a set of deals, D, consistent with the bidding thus far, and then for each bid b from our candidate set, we use the database to predict how other players will bid and thus how the bidding will conclude, and then compute the double dummy result of the contract that was reached. We then select the bid b with the best score with regard to the double dummy result.

One important aspect of bidding in bridge is that every bid has an established meaning. There

are a variety of different rules, or conventions, for decyphering the meaning of a bid, but players in a bridge game must not only understand their partners' conventions, they must also be able to provide these same conventions to their opponents upon request. Thus, in order to successfully interact with human bridge players, a computer player needs to understand a wide variety of different human conventions. This also complicates the creation of any sort of "machine-optimized" bidding process, since bids used by the computer player must be explainable to human opponents in plain, natural language.

Now we turn to the manner in which GIB generates its hypothetical card deals for the Monte Carlo simulations. To do this, Ginsberg uses a two-stage process. First, he produces a set of candidate deals that are consistent with the bidding of all players. This is done by first producing constraints on the number of cards in each suit held by each player based on the established meaning of the bids according to the bidding database, and then validating each deal by asking GIB's bidding module if it would have bid in the same way the player was observed to have done while holding that hand. Deals that pass this test move on to the second stage, where the card play of the deal is analyzed. In this stage, GIB evaluates the probability of whether a player is holding a particular card C based on whether the program's search algorithm indicates that the given player *should* have played C if indeed she possessed it. Defensive signalling, such as following standard conventions so as to convey information to one's partner, is also taken into account, resulting on a probabilistic weight w_d attached to each deal d. w_d is then used to weight the results of the alpha-beta search performed on d.

3.1.3 The Best Defense Model and its use in Bridge

In 1998, Frank and Basin published an extensive critique of the Perfect Information Monte Carlo search (or as they call it, repeated minimaxing) approach to imperfect information games [9]. In so doing, they identified and formalized the sorts of mistakes that PIMC search is guaranteed to make through two distinct concepts that they term strategy fusion and non-locality. We will discuss these concepts in much greater detail in Chapter 5. In this work, Frank and Basin also propose a new model of analysis for imperfect information games. One of the major goals of the authors is to reconcile the formal model used by computer scientists for analyzing bridge and other imperfect information games with the model implicitly used by human experts in bridge literature. The "Best Defense" model is the result of this ambition.

Building the Best Defense model of an imperfect information game is based on three major assumptions. The first is that we assume our opponent (henceforth, *the minimizer*) has perfect information of the game whereas we (*the maximizer*) do not. Secondly, we assume that the minimizer chooses her strategy after the maximizer. And thirdly, we assume that the maximizer adopts a pure, rather than a mixed, strategy. Not only do these assumptions coincide with those made in expert bridge literature, but they result in a formal model that is amenable to analysis. This results in an al-

gorithm for solving the best defense form of a game which Frank and Basin term *exhaustive strategy minimisation*. The algorithm enumerates all possible strategies of the maximizer, and a minimisation operation is performed on each such strategy. This algorithm, however, is doubly exponential in the number of maximizing layers in the game tree, and thus Frank and Basin contend that it is not entirely intended for practical use, but as a framework through which other architectures can be analyzed.

In 2000, Frank et al. produced Finesse, a system capable of optimally solving single-suit bridge problems — a subset of the full game of bridge in which only the 13 cards in a given suit are considered [10]. Finesse solves these situations by searching over a space of expert-defined tactics, rather than over the space of individual card plays. Since this abstract space is much smaller than the original state space, Finesse can actually solve the single-suit deals under Frank and Basin's Best Defense model — in other words, by enumerating the maximizer's strategy space and computing a complete set of payoff vectors obtained by each strategy. The authors demonstrate results on an authoratative set of 1500 single-suit bridge problems for which the abstractions used by Finesse do not cause it to miss any optimal plays, and indeed the program discovers a 3% error rate in the solutions given by experts.

Of perhaps equal interest to Frank et al. is Finesse's ability to explain its play in human readable terms. These explanations can be easily extracted from Finesse's game-tree, since the tree is already built over abstract strategies. The authors show that the explanations produced by Finesse closely match corresponding explanations by human experts, and even express their intent for Finesse to author a bridge textbook of its own. Unfortunately, however, solving single-suit card configurations is a long way from playing the full game of bridge complete with bidding, and it is quite likely that the full game is simply too large for an approach based on the Best Defense model.

From a theoretical standpoint, the Best Defense model is more satisfying than PIMC search in the sense that whenever a guaranteed winning move is found in the model, it is also guaranteed to be a winning move in the original imperfect information game. This property does not hold for PIMC search, as Frank and Basin have shown. However, a drawback that the Best Defense model shares with PIMC search is that it makes no allowance for exploiting an opponent's ignorance. It is very possible that under Best Defense, some move M is reported as being a guaranteed loss because the opponent is assumed to be playing perfectly with perfect information, whereas in reality, M may have a very high chance of success due to the very fact that our opponent does *not* have perfect information. In some sense, we may say that exhaustive strategy minimisation in the Best Defense model is sound, but not complete: any move which Best Defense identifies as good is actually good, but there may be many actually good moves which are discarded as bad under the assumption of Best Defense.

3.1.4 Current Techniques in Computer Bridge

Since Ginsberg's introduction of GIB, there has been relatively little published work on computer players that play the game of bridge in its entirety. Sub-problems of the game have, however, been examined by some authors. Mossakowski and Mandziuk [29] describe the use of neural networks to predict the outcome of perfect information (or double-dummy) bridge problems and report a level of accuracy comparable to human experts on some classes of deals. Amit and Markovitch [1] devised a learning system for bridge bidding, with particular emphasis on training cooperative agents simultaneously. This approach is particularly salient in bridge, since in most bridge tournaments, participants enter in pairs and will play with the same partner for the entire tournament. Amit and Markovitch showed that their program could outperform GIB's bidding module. Interesting and challenging problems not directly related to constructing strong computer bridge players have also appeared in the literature. Kemp et al. [20] discuss the techniques needed to construct an effective automated bridge tutoring system, and Yan [49] discusses the problem of detecting illegal collusion (such as partners sharing hand information over external channels) in online bridge play. Computer bridge competitions, such as the World Computer Bridge Championship, continue to be held annually, and the strongest programs are said to still be based on Ginsberg's GIB [35].

3.2 Skat

3.2.1 A Double-Dummy Solver for Skat

Although commercial computer skat programs do exist, the first published work of which we are aware that relates to skat was by Kupferschmid and Helmert in 2007 [23]. These authors applied Monte Carlo simulation with alpha-beta search to the game of skat to produce a player they simply call Double-Dummy Solver, or *DDS*. DDS is largely constructed using a straightforward implementation of the Perfect Information Monte Carlo algorithm, but with two major enhancements to the alpha-beta search component. The first, *quasi-symmetry reduction*, is an adaptation of Ginsberg's partition search that groups together search states that are "nearly" equivalent. For example, in skat, a player may hold two cards that are adjacent in terms of rank, or *rank equivalent*, but which differ in point value. Therefore, playing either card results in states that are strategically equivalent in terms of the number of tricks won, but which differ in end result by at most the difference in point value between the two cards. This observation can be used to bound the alpha-beta search, and therefore prune the game tree. The second enhancement used by Kupferschmid and Helmert, *adversarial heuristics*, calculates guaranteed upper and lower bounds on the number of points that can be won by the declarer, in a manner somewhat similar to heuristics used in single-agent search problems, and uses these bounds to prune sections of the game tree.

Kupferschmid and Helmert show that the resulting DDS system can solve individual double-dummy deals quickly. They also provide a small indication of the program's actual playing strength,

stating that it defeated the expert system XSkat (available at `www.xskat.de`) 17 out of 18 games, and performed comparably with human players of unspecified caliber on a similar number of hands. The authors do not specify how DDS generates hypothetical deals of the cards that are used in the repeated alpha-beta searches, or whether weighted Monte Carlo samples are used as was the case for Ginsberg's GIB.

Although in principle, DDS could also use Monte Carlo simulations for bidding, Kupferschmid et al. instead turn to a clustering method based on the *k-nearest neighbors* classification algorithm [19]. K-nearest neighbors is a simple machine learning algorithm in which a new, unlabelled example x is assigned a label by comparing it to a set of labelled exemplars; in particular, the k exemplars that are closest to x by virtue of some distance metric. Kupferschmid's system is in fact a collection of 5 different classifiers, one for each different game type in skat. When classifying an unknown sample x, for each game type t the system assigns x a score in the interval $0 \ldots 120$ in terms of the number of card points the declarer is expected to win if he were to play a game of type t. This is done by computing a numeric feature vector $f(x)$ and using Euclidean distance to find the k nearest neighbors from among the system's knowledge base. The value assigned to the hand is then simply the average of the values of these k neighbors, weighted by the relative distance of each neighbor. Kupferschmid et al. produce the system's knowledge base by evaluating several thousand card hands using DDS's alpha-beta search procedure, and reports a final accuracy of 75% in terms of correctly classifying 'winnable' games. No results are given however on actual games played using this bidding system, so the overall playing strength of DDS with k-nearest neighbor bidding was previously unknown. We provide some such results later in this document.

3.2.2 The UCT Algorithm

The Upper Confidence bounds on Trees, or *UCT*, algorithm by Kocsis and Szepesvari [21] is a very different approach to planning in games from the traditional minimax search approach, and has been applied to Skat in work by Schäfer [38]. UCT is effectively a sampling algorithm that keeps statistics on the performance of different moves at each node of the game tree. The algorithm performs numerous simulations, where each simulation corresponds to a single play of the entire game, and where the move at each game tree node N is selected according to a formula of the form:

$$M = argmax_i(X_i + C\sqrt{\frac{2\log t_i}{T}}) \tag{3.1}$$

where X_i is the average result of selecting move i so far at N, t_i is the number of times move i has been selected, and T is the total number of times that N has been visited during the search.

This selection method provides a balance of exploration and exploitation — the first term of equation 3.1 encourages "promising" moves to be selected most of the time, while the second ensures that all moves will eventually be examined and the simulations will never completely stop exploring the game tree. The algorithm can be terminated after any number of iterations, at which

point we simply play the move with the highest average score at the root node of the search.

In practice, memory constraints will usually make it infeasible to keep statistics on every node in the game tree. Therefore, it is common to start maintaining statistics at a game tree node N only after N has been visited a sufficient number of times during the simulations performed by UCT. In this case, however, UCT still requires some form of playout module that is capable of selecting moves statically at nodes where statistics are not yet being kept. This playout module could be as simple as selecting moves at uniform random, or could be elaborately constructed from expert domain knowledge.

The UCT algorithm first showed its major promise in the game of go [48]; it is probably fair to say that UCT has formed at least the original basis and inspiration of most major computer go players, including all the top players at the 2008 Computer Olympiad. UCT has also been successful in the General Game Playing competition, first appearing in the competition and winning in 2007 [8]. Since then, most of the strongest players in GGP have used UCT, including the winner from 2009 and 2010 [28]. UCT has also been implemented in the game of skat by Schäfer [38]. To construct his system, Schäfer used XSkat, the best known rules-based skat program, as his playout module for completing games at the leaf nodes of his UCT tree, and simply created hypothetical deals at uniform random for use in the UCT simulations. Schäfer then showed that the resulting UCT player significantly defeated XSkat playing on its own, although his results were inconclusive against a PIMC search-based player.

3.3 Poker

Poker (and in particular, Texas Hold'em poker) is another game which has recently seen the appearance of successful computer players. Since poker crucially depends on the assumption that neither player knows for sure (except in rare cases) which one holds the better hand, a Monte Carlo search approach based on perfect information search seems likely to be doomed to failure. Rather, in poker it is important to be able to employ a randomized or *mixed* strategy, such that the opponent cannot deduce our cards based on our play.

A recent but successful approach to achieving this behavior is the use of *counterfactual regret minimisation*, or CFR, as proposed by Zinkevich et al. [18] and implemented in poker by Johanson [16]. Unlike Monte Carlo search and UCT, CFR is an offline process that learns move probabilities at every information set of the game. The algorithm is an iterative one; each iteration consists of a run through the game tree with the move at each information set being selected according to the current move probabilities stored at that node. Upon reaching a terminal state of the game, every node in the tree is updated according to how much the program regrets playing the strategy that was actually played compared to best available move at each information set. This process will converge to a Nash equilibrium in the two-player case, although in practice this can take a considerable amount of time to occur. However, the resulting computer player is simply an enormous table of move probabilities,

meaning that nothing more than a table look-up needs to be performed when the program is actually playing games.

CFR improves considerably over previous methods for approximating Nash equilibria in that it requires memory linear in the number of information sets, as opposed to actual game states. However, for full-scale, large games such as poker (with 10^{18} states) or, bigger yet, skat, this is unfortunately still not feasible. To combat this drawback, Johanson instead runs the CFR algorithm over a smaller, *abstract* version of the game of poker. For instance, instead of considering individual permutations of cards, Johanson creates buckets that group together sets of cards that have similar strategic properties, notably in terms of the strength of the hand in terms of its probability of defeating the opponent. Under this scheme, strategies can then be defined over sequences of a small number of buckets, instead of over sequences of actual cards, making the game small enough to be amenable to CFR.

This general approach of finding a Nash equilibrium in a smaller, abstract game and mapping this strategy back to the original game has been quite successful in computer poker, and has been used by several of the top computer players [50] [14]. Beyond building static, equilibrium strategies, poker AI researchers have also been concerned with the problem of opponent modelling — that is to say, dynamically exploiting observed weaknesses of an opponent's strategy. When research in this field first began in earnest in 2001, the critical need for opponent modelling was cited as a key feature that distinguished poker from more traditional perfect information games [2]. However, although early forays into poker opponent modelling do exist [7] [42], an examination of the top contenders of the 2010 AAAI Computer Poker competition (`http://www.computerpokercompetition.org/`) shows that the clear majority are static players with little online adaptation to a particular opponent.

There has, however, been some recent exciting progress on this front. In 2009, Johanson and Bowling [17] proposed Data Biased Response (DBR), an opponent modelling technique that balances an agent's ability to exploit an opponent while avoiding being exploited itself. Most impressively, DBR comes with theoretical guarantees that the trade-off it achieves between opponent-exploitation and self-exploitability is optimal. However, there are two caveats to this approach: first, DBR still requires on the order of thousands of observations to begin significantly exploiting the opponent, and second, computing the DBR strategy itself once given its observations is much too slow to be done in real time. In 2011, Ganzfried and Sandholm proposed a method that lacks some of the guarantees of DBR, but that can function in real-time and using only in-game observations of the opponent's behaviour (for example, their approach does not get to observe the opponent's cards in situations that did not lead to a showdown) [13].

3.4 Scrabble

Scrabble is another imperfect information game that has seen the appearance of strong computer players. The first program to seriously challenge (and ultimately defeat) human experts was the

program Maven by Sheppard [40]. Although much of the challenge in Scrabble stems from the need for fast move generation, Maven also uses simulation roll-outs of multiple hypothetical worlds to evaluate positions in a manner somewhat similar to Ginsberg's GIB. More recent work by Richards and Amir [32] highlights Scrabble's imperfect information aspects. In this work, Richards and Amir enhance Quackle, at that time the world's strongest Scrabble program, with an inference component to bias hidden information in the game (in particular, letter distributions for the opponent's rack) for use in simulations. This was done using Quackle's static move evaluator in conjunction with Bayes' Rule to calculate the probability that the opponent would have played as observed given different hypothetical letter racks. The addition of this inference allowed Quackle to defeat its prior incarnation by a significant margin.

3.5 Kriegspiel Chess

Kriegspiel Chess is an imperfect information chess variant in which a player sees only the positions of his own pieces. When a move is attempted, an impartial and omniscient referee informs the player of whether or not the move is in fact legal. The first known computer player to play kriegpsiel chess in its entirety at a reasonable level is by Parker et al. [31]. The basic approach used by this program is similar to the PIMC search used by Ginsberg in bridge. A key challenge in kriegspiel chess is that, unlike in bridge, skat and poker, merely generating hypothetical game states that are consistent with the game's history can take exponential time. Parker et al. overcome this by considering only hypothetical states that are consistent with the most recent set of observations in the middle to late game, and show that with reasonable probability, such states will also be consistent with the entire history. Later, Ciancarini et al. [5] created a program they call Darkboard, which groups together large numbers of states into *metapositions*. Using this technique, Darkboard defeated the program by Parker et al.. More recently, Ciancarini and Favini have applied the UCT algorithm to kriegspiel chess with good results [6]. However, as kriegspiel chess is not nearly as popular among humans as the previously discussed games in this chapter, it is difficult to assess the strength of kriegspiel playing programs in more absolute terms.

3.6 Hearts and Issues in Multi-Player Games

Hearts is a card game for four players in which each player endeavors to take as *few* points as possible. It is similar to both bridge and skat in that it is a trick-taking card game, but in some ways simpler, as there is no auction phase nor a notion of multiple game types. It is, however, complicated by the fact it is a multi-player game, without the permanent alliance between players in bridge, or even the temporary alliance formed between the defenders during the cardplay phase of skat.

One of the strongest known hearts programs was created by Sturtevant and White [45]. According to these authors, search-based programs have generally been very weak in hearts due to

the game's multi-player nature. Instead, Sturtevant and White used the TD(λ) algorithm, one of the most common techniques in reinforcement learning, to construct a hearts player that combines learning and search. Linear regression is used over a set of features to abstract the state space for learning, after which the system simply learns through self-play and by playing against a competing search-based player. An interesting note about this work is that the learned player only learns to play the perfect information version of Hearts, and is then simply used as a module for a top-level Monte Carlo search in order to play the real, imperfect information game, similar to Ginsberg's GIB. The resulting player is shown to significantly defeat one of the strongest known pure search-based players. Later, Sturtevant applied the UCT algorithm discussed in Section 3.2.2 to the hearts domain with strong results [44].

Sturtevant's work in hearts is both interesting and relevant because both trick-taking card games previously mentioned in this thesis — bridge and skat — are in actual fact multi-player games. In the literature, bridge has typically been treated as a two-player game, but in actual fact it is played by four different players; the abstraction to a two-player game makes sense because the players are allied in teams of two, and all existing approaches to the game have been based on perfect information models of the game where we ignore the fact that two partners do not necessarily know each others' hand, and therefore may not actually play as though they were a single entity. Skat stradles the multi-player line even more precariously than bridge; although the cardplay phase can (and has been) treated as a two-player game, during the auction phase, each player is out for herself. Thus, in studying skat, we must be conscious of some of the pitfalls found in multi-player games. Sturtevant has previously outlined some of these pitfalls [43], particularly the need for strong opponent modelling in multi-player games. As mentioned in our game theoretic background in Chapter 2, this is primarily due to the fact that equilibrium strategies are not interchangeable in multi-player games. In other words, no "safe" opponent-invariant strategy exists in this domain in the general case. Human skat players are certainly acutely aware of this issue: e.g. the word "Maurer" (or mason) is a widely used term among skat players that describes a player who intentionally holds back during the bidding process so as to defeat the eventual soloist with an unexpectedly strong defender hand. What's more, Von Stengel and Koller [47] have shown that in non-cooperative, zero-sum team games, in which a team of agents with shared payoffs but disparate information competes against an omniscient adversary, the best payoff that the team players can hope for is given by the *team-maxmin equilibria*. Von Stengel and Koller show that a team-maxmin equilibrium is distinct from the equilibrium that would arise in the 2-player version of the same game, and furthermore the payoff to the team can be lower than in the case where the team is treated as a single player. Since in skat the two allied defenders are typically less informed than the soloist (because the soloist knows the two cards hidden in the skat), this result by von Stengel and Koller indicates that any approach treating skat as a 2-player game even in the cardplay phase may be unable to capture the optimal strategy of the true multi-player game.

Time's fun when you're having flies.

-Kermit the Frog

4

Kermit: An Expert-Level Computer Skat Player

In this chapter, we will describe the inner workings of Kermit, a computer player for the game of skat capable of playing the game in its entirety. Our goal for Kermit was not only to produce a strong player capable of challenging both other programs and humans alike, but also to establish a baseline for our future experiments and investigations. Many of the techniques used to build Kermit — particularly the Perfect Information Monte Carlo search used for its cardplay — are not novel on their own. Rather, the main contribution highlighted by Kermit is the synthesis of effective techniques to create a strong computer skat player that plays at human speeds.

We will separate our discussion of Kermit into three major topics. First is the search mechanism that Kermit uses to select moves during cardplay. Next is the bidding phase, in which we primarily employ a learned static state evaluator to assess soloist winning potential for individual hands. Third, we will discuss the inference procedure that Kermit uses to make more informed guesses about hidden information in the game (e.g. the cards held by the opponents).

Finally, the author wishes to make it clear that he is by no means the sole contributor to the construction of Kermit and the necessary surrounding skat infrastructure. In particular, credit for implementing Kermit's perfect information alpha-beta search and the discovery and fine-tuning of table features belongs to Dr. Michael Buro; many basic skat data structures were written by Nathan Taylor.

4.1 Perfect Information Monte Carlo Search for Cardplay

We have already briefly discussed the Perfect Information Monte Carlo (PIMC) search technique in Chapter 3. This is the algorithm that Kermit uses to select moves during cardplay, and so we will revisit it here. Recall that PIMC search works by repeatedly sampling fully observable worlds and calculates the value of each of our available legal actions in each such sampled world. In the end, we simply select the action with the highest accumulated score over all worlds. Pseudocode for this procedure is shown in Algorithm 1. If time permits, then conceivably every possible world could be sampled; in practice, one will normally sample as many worlds as possible within the allotted time-frame. This process can easily be parallelized, in the simplest case by essentially running two separate instances of the PIMC search algorithm and in the end aggregating the final move values (although this can result in duplicated work if the same world is sampled in both processes). This

Algorithm 1 Pseudocode for Perfect Information Monte Carlo Search

Require: current state s, set of worlds W that are legal in state s, maximum available time T
  for each action a that is possible at s **do**
    {Initialize values for all actions}
    $v(a) \leftarrow 0$
  end for
  $t \leftarrow elapsedTime$
  while $t < T$ **do**
    sample some world $w \in W$
    for each action a that is possible at s **do**
      {applyAction(w, a) returns the world that results when action a is applied to world w}
      $w' \leftarrow applyAction(w, a)$
      $v(a) \leftarrow v(a) + solve(w')$
    end for
    $t \leftarrow elapsedTime$
  end while
  return $argmax_a(v(a))$

is the approach that Kermit takes. Typically, Kermit can examine approximately 40 to 50 worlds per process in the first trick of a skat hand, and can examine every remaining legal world by the mid-game.

PIMC search owes much of its strength to the computational tractability it gains by considering different worlds to be completely independent, i.e. that the best move in some world $w \in W$ depends only on w itself, and not on any other $w' \in W$ for which $w \neq w'$. This strength is also its weakness, since the independence assumption leads to the theoretical deficiencies identified by Frank and Basin as mentioned in Chapter 3, but we will defer a deeper discusion of these issues to Chapter 5.

The key operation in the PIMC search procedure is $solve(w)$, which attaches a game value to a perfect information world w. In principle, any form of evaluator, be it exact or heuristic, could be used here. For Kermit, as per Ginsberg's previous work in bridge, we use an alpha-beta minimax search to compute perfect information game values. This, of course, requires the capability to solve perfect information skat deals efficiently, which we describe further below.

4.1.1 Perfect Information Search Enhancements

Kermit uses standard alpha-beta minimax search to compute the value of perfect information worlds. Since skat uses a smaller deck than other games such as bridge, this computation is relatively straightforward. Nevertheless, for the game's first trick, the time required to solve a single world is not insubstantial. Here we describe the search enhancements used to speed up Kermit's alpha-beta search.

Kermit makes use of standard alpha-beta search enhancements such as transposition tables and shallow searches for sorting moves close to the root. For sorting moves in interior nodes, we combine numerous general and game-specific heuristics. Notably, the addition of the fastest-cut-first search heuristic [11], which rewards moves based on their beta-cutoff potential and the estimated size of the

underlying subtree, reduced the search effort by approximately 40%. The idea is that in cut-nodes it is sufficient to find one move with value $\geq \beta$. Therefore, in inhomogeneous trees visiting moves with high cutoff potential and small size first may lead to better search performance than starting with the potentially best move.

We also implemented a series of card-game- and skat-specific move ordering heuristics. As in [23], we group cards according to their strength and only search one representative move in each group. For instance, when holding 7 8 suited in one's hand, both are equivalent. Similarly, 7 and 9 are equivalent if the 8 has been played in a previous trick. Care must be taken not to consider current trick-winning cards as being already played, because the player to move can still decide to take the trick or not. For example, when holding 79 in a suit in which the 8 has just been led, 7 and 9 are not equivalent. This sound forward pruning heuristic applies to all cards in null-games and the 789 and Jack card groups in trump games. In addition, at the beginning of the game we group Queens with Kings and Tens with Aces at the cost of small score differences. Our remaining heuristics for trump games reward moves for winning tricks, catching high cards, and putting the soloist in the middle position of play. In null-games, we bias move selection towards avoiding the soloist's void suits, preferring the co-defender's void suits, and sloughing the highest card in unsafe suits.

4.1.2 Search Payoffs

The payoff at the leaf nodes of Kermit's alpha-beta search tree is the *card point* differential between the two competing parties, the soloist and the defenders. Recall that the total number of card points in the game is 120 and the soloist needs 61 of these to win. For the purpose of these perfect information searches, the two defenders are treated as a single player in the search.

When searching an individual perfect information world, Kermit does not in fact perform only a single search. Rather, we employ a series of *zero-window* searches to narrow down the search space. A zero-window alpha-beta search is one in which alpha and beta are set to the values v and $v + 1$ respectively. Performing a search with such a window will determine whether the true value of the position is greater than or less-than-or-equal-to v. We begin with a zero-window search to determine whether the position is a win or a loss for the soloist. Based on the outcome of this search, we perform a second zero-window search to determine whether the winning side can play the loser "schneider," which requires taking 90 or more of the card points and results in an additional game value multiplier for the soloist. Finally, we widen the search window given the range determined by the previous two searches and determine the actual card points that are taken by each side. The final value that is returned by the search prioritizes winning the game, with the final card points won added on as a small "tie-breaker" to distinguish between positions with the same win/loss value.

Note that this final stage of the search could potentially be omitted, since actual card points won make no difference to the game's final value (e.g. a soloist win with 61 card points is just as good as one with 85 in terms of his list-score). Doing so would speed up the search process,

allowing for more hypothetical worlds to be examined. However, empirically, we have found this final optimization to be worth its cost in terms of playing strength.

The null game type, where the soloist's goal is to lose *every* trick, does not quite fit into the framework above, since card points are irrelevant in this context. Therefore, for null games, Kermit searches primarily for win/loss distinction, with a small bonus for node-depth for the soloist (and therefore a penalty for node-depth for the defenders). This latter adjustment causes Kermit, as soloist, to favour positions that prolong the game as much as possible when it thinks it is losing. This adjustment has been critical for Kermit's null-game performance, as otherwise there appear to be too many positions that are considered a certain loss under the perfect information assumption with no way to distinguish between them.

4.2 State Evaluation for Bidding

When solving a game is infeasible due to large state spaces, it becomes necessary to approximate state values. There exist large bodies of literature in statistics, machine learning, and AI that cover the design, optimization, and evaluation of function approximators which can be used to evaluate the utility of a state for a player. Classical examples are linear material-based chess evaluators and artificial neural networks trained to predict the outcome of backgammon games [46]. Function approximators have been mostly trained for two-player perfect information games, although there has been some work on learning evaluations in multi-player card games as well. Sturtevant and White [45] describe a system that successfully uses TD-learning and a mechanism to generate features to learn a linear evaluation function for the perfect information version of hearts.

In the construction of Kermit, we propose approximating imperfect information state values directly and using such estimates in game tree search for the bidding phase of the game. If imperfect information game data is available, evaluation parameters can be estimated directly using supervised learning techniques. Otherwise, it may be possible to bootstrap imperfect information evaluations from perfect information evaluations and from self-play using cardplay methods such as PIMC search.

In principle, the same PIMC search used for cardplay could be used to evaluate positions for bidding potential as well. For example, if for a particular hand a PIMC search beginning from the play of the first trick finds that the player to move wins in 75% of the worlds it examines and loses in the remaining 25%, this information could be used to make bidding decisions. In Kermit, we do not take this approach for two reasons. The first is speed. After the soloist has won the bid, she will often pick up the two cards in the skat, where 231 different possible card combinations await. She then discards two cards, for which she has 66 possibilities, and then must decide between six major game types, resulting in a final factor of 91476. It would be infeasible for Kermit to perform a first-trick PIMC search for even a small fraction of this space and still play at human speeds. The second, and more important, reason is that while PIMC search appears to perform well at differentiating moves

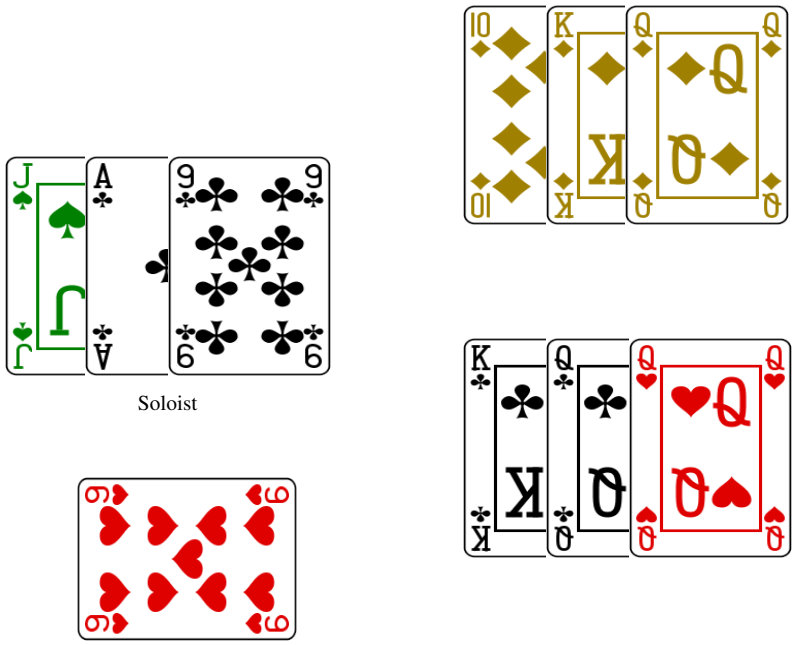


Figure 4.1: A miniature skat position where PIMC search will underestimate the soloist's winning chances.

during cardplay, it is much less accurate at predicting actual game outcomes. We will illustrate this seemingly contradictory phenomenon by means of an example.

Consider the miniature deal shown in Figure 4.1. We will say that this is a 'Grand' game, where only the ♠J held by the soloist, is trump. The soloist is to lead and needs to take all the remaining card points to win. Let us also say that the soloist has discarded the ♡9 but that from the defenders' perspective, it is possible that the ♣9 was discarded instead. A PIMC search from the soloist's perspective in this setting will correctly identify that playing the ♣9 first is a terrible mistake; it is much better to play the ♣A first, resulting in a winning position in the event that the ♣K and ♣Q are split between the two defender hands. However, in the real game, the situation where the ♣K and ♣Q are *not* split is not a guaranteed loss for the soloist (although PIMC search will believe that to be the case). When the soloist plays the ♠J, it is possible that one of the defenders will have to choose between keeping the ♣Q and the ♡Q, and will only make the right decision 50% of the time. Therefore, PIMC search will not correctly assess the soloist's winning chances in this scenario, but will generally play the cards correctly if forced to do so.

Scenarios of this sort appear to be common in skat. Table 4.1 shows the result of estimating the soloist's winning chances by performing a PIMC search at the beginning of the cardplay phase on a collection of 10,000 games played by humans, separated by game type. This is compared to the actual soloist win rates based on the outcomes of these same games. We see that PIMC search significantly underestimates the soloist's winning chances in all game types, with this effect being most severe in null games. In light of this, we turn to the learned static evaluations that we describe

Table 4.1: Soloist win% estimation as calculated by PIMC search as compared to empirical soloist win-rates on a collection of 10,000 human games.

Game Type	PIMC Search Win%	Actual Win%
Suit games	67%	80%
Grand	82%	93%
Null	58%	75%

in the next section.

4.2.1 Learning Table-Based State Evaluations

When applying learning to game tree search, evaluation accuracy and speed are a concern, because good results can be obtained either by shallow well-informed search or deeper but less-informed search. In trick-based card games many relevant evaluation features can be expressed as operations on card groups such as suits or ranks. Therefore, the generalized linear evaluation model (GLEM) framework [3], on which world-champion caliber Othello programs are based, is suitable for creating highly expressive yet efficient evaluation functions for card games. The top-level component of GLEM implements a generalized linear model of the form

$$e(s) = l\big(\sum_i w_i \cdot f_i(s)\big),$$

where $e(s)$ is the state evaluation, l is an increasing and differentiable link function, $w_i \in \mathbb{R}$ are weights and $f_i(s)$ are state features. For the common choices of $l(x) = x$ (linear regression) and $l(x) = 1/(1 + \exp(-x))$ (logistic regression) the unique parameters w_i can be estimated quickly, but the resulting expressiveness of e may be low. To offset this limitation, GLEM uses table-based features of the form

$$f(s) = T[h_1(s)] \ldots [h_n(s)],$$

where index functions $h_j : s \mapsto \{0, .., n_j - 1\}$ evaluate properties of state s and the vector of indexes is used to retrieve values from multi-dimensional table T. The advantages of table-based features are that table values can be easily learned from labeled samples and state evaluation is fast.

4.2.2 Application to Skat Bidding

Prior to our work on Kermit, bidding was often considered as a particular weakness of skat programs. In commercial programs, bidding is believed to be commonly implemented using rule-based systems designed by program authors who may not be game experts. We have previously mentioned the work by Kupferschmid et al. describing a bidding system based on k-nearest-neighbor classification which estimates the card point score based on the soloist's hand and a knowledge base of hand instances labeled by DDS game results. [19]. We will later compare our work against their approach.

Our bidding system is based on evaluating the soloist's hand in conjunction with the cards that have been discarded and the type of game to be played (we call this the "10+2 evaluation"), estimating winning probabilities by means of logistic regression. We believe that basing the strength of

hands on winning probability or expected payoff, rather than expected card points as Kupferschmid et al. had done, is more suitable because the soloist's payoff in skat mostly depends on winning the game, e.g. winning with 61 or 75 card points makes no difference. We use a set of table-based features which are more expressive than the mostly count-based features used in previous work. Lastly, we have access to 22 million skat games that were played by human players on an Internet skat server located in Germany. This allows us to mitigate the problems caused by DDS-generated data and to benefit from human skat playing expertise which is still regarded as superior when compared to existing skat programs. No filtering of this data was performed however, and players of a wide range of skill are represented in the data set.

Our 10+2 evaluation $e_g(h, s, p)$ estimates the winning probability of the soloist playing game type $g \in \{\text{grand}, \clubsuit, \spadesuit, \heartsuit, \diamondsuit, \text{null}, \text{null-ouvert}\}$ with ten-card hand h in playing position $p \in \{0, 1, 2\}$ having discarded skat s. We describe the features used to build this estimator in the following subsections.

Null-Game Evaluation

Conceptually, the null-game evaluation is the simplest. It is based on estimating the strength of card configurations in h separately for each suit. Recall that the soloist wins null-games if he doesn't take a trick. Certain card configurations within a suit are safe regardless of whether the soloist has to lead or not (e.g. ♣789). Others are only safe if the soloist doesn't have to lead (e.g. ♣79J). This suggests creating a table that assigns to each of the $2^8 = 256$ suited card configurations a value representing the safety of that configuration. We define this value as the winning probability under the assumption that there is only one weakness in h. To evaluate h we then simply multiply the four associated configuration values assuming independence. Table values can be easily estimated from human training data. It is hard to estimate the true winning probability of unsafe null-games using DDS because the defenders get to see the weakness in h and exploit them perfectly. This leads to DDS almost always predicting soloist losses. Taking the defenders' ignorance into account, however, the soloist often has a considerable winning chance (e.g. having a singleton Ten and a void suit), which is reflected in the table values learned from human data.

Trump-Game Evaluation

Recall that in skat trump-games the soloist needs to get 61 or more card points to win. Because Aces and Tens are worth the most by far, skat evaluation functions need to be able to judge which party will get these cards over the course of the game. The soloist can obtain high-valued cards in several ways: by discarding them in the skat, trumping high cards played by the defenders, playing an Ace and hoping it will not be trumped, or trying to catch a Ten with an Ace.

Good skat players understand the trade-off between the number of tricks won and the number of guaranteed high cards. Ron Link, who is one of North America's strongest skat players, has

developed the following simple heuristic which helps him making move decisions in all game phases (personal communication, 2008): "If you can secure k high cards, then you can give away $k+1$ tricks and still be confident of winning". Our evaluation functions for suit- and grand-games are based on the GLEM evaluation framework. They implement the above evaluation idea using tables to evaluate the expected number of points and tricks obtained during play. High cards for the trump suit and each off-trump suit are evaluated independently and both are combined by means of logistic regression, i.e.:

$$e_g(h,s,p) = 1/(1+\exp(w_0^g + \sum_{i=1}^{4} w_i^g \cdot f_i^g(h,s,p))),$$

where f_1^g, f_2^g and f_3^g, f_4^g evaluate the number of card points and the number of tricks and high-valued cards made in the trump suit and off-trump suits, respectively, for trump game type g, and $w_j^g \in \mathbb{R}$ are maximum-likelihood weight estimates computed from a set of labeled sample positions.

To compute the features $f_1^g, \ldots, f_4^g$, we employ a table-based evaluation using the primitive index features described below. These table index features can be computed quickly by bit operations on words of length 32 (representing card sets) and, when combined, form a set of expressive non-linear features that capture important aspects of skat hands correlated with winning trump-games:

- $\{\clubsuit^\star, \spadesuit^\star, \heartsuit^\star, \diamondsuit^\star\}(c) \in \{0, \ldots, 2^7-1\}$ gives the index corresponding to the configuration of cards in c, restricted to a given suit, excluding Jacks. E.g. $\clubsuit^\star(\clubsuit 78 J \heartsuit JA) = 2^0 + 2^1 = 3$.
- $\{\overline{\clubsuit^\star}, \overline{\spadesuit^\star}, \overline{\heartsuit^\star}, \overline{\diamondsuit^\star}\}(c) \in \{0, \ldots, 2^{11}-1\}$ is the same as $\{\clubsuit^\star, \spadesuit^\star, \heartsuit^\star, \diamondsuit^\star\}(c)$, but includes all Jacks. E.g. $\overline{\clubsuit^\star}(\clubsuit 78 J \heartsuit JA) = 2^0 + 2^1 + 2^{10} + 2^8 = 1283$.
- $\mathrm{tc}(t,c) \in \{0,1\}$ distinguishes two *trump contexts*: 0 if the # of cards in c from trump suit t is ≤ 5, 1 otherwise. I.e. hard-to-win hands.
- $\mathrm{tt}(t,o) \in \{0,1,2,3\}$ counts high-valued *trump targets* (Aces and Tens) amongst opponents' cards $o = (h \cup s)^c$ which are not from trump-suit t. The count is clipped at 3, as usually no more than 3 high cards can be trumped.
- $\mathrm{vt}(t,h) \in \{0,1,2\}$ represents the number of high cards in the soloist's hand that are *vulnerable* to defenders' trumps (clipped at 2) — for trump suit t.

Using these building blocks, our table-based suit-game evaluation features are defined as follows (assuming, for example, that $\clubsuit$ is trump):

$$f_1^\clubsuit(h,s,p) = \sum_{x \in \{\spadesuit^\star, \heartsuit^\star, \diamondsuit^\star\}} \mathrm{sidePoints}\big[\mathrm{tc}(\clubsuit, h \cup s)\big]\big[x(s)\big]\big[x(h)\big]$$

$$f_2^\clubsuit(h,s,p) = \sum_{x \in \{\spadesuit^\star, \heartsuit^\star, \diamondsuit^\star\}} \mathrm{sideTricks}\big[\mathrm{tc}(\clubsuit, h \cup s)\big]\big[x(s)\big]\big[x(h)\big]$$

$$f_3^\clubsuit(h,s,p) = \mathrm{trumpPoints}\big[\overline{\clubsuit^\star}(h)\big]\big[\mathrm{tt}(\clubsuit, (h \cup s)^c)\big]\big[\mathrm{vt}(\clubsuit, h)\big]$$

$$f_4^{\clubsuit}(h,s,p) = \text{trumpTricks}\big[\overline{\clubsuit^\star}(h)\big]\big[\text{tt}(\clubsuit,(h\cup s)^c)\big]\big[\text{vt}(\clubsuit,h)\big]$$

The table entries of the features above predict how many card points will be taken (sidePoints) and how many tricks and high cards are secured (sideTricks) in the course of a game, given that the soloist holds cards h and the skat contains s. The side-suit tables each consist of $2\cdot 128\cdot 128 = 32768$ entries, of which only 4374 are actually in use because our encoding is not tight. The *trump context* is used to help distinguish between hard to win scenarios where the soloist has fewer than 6 trump cards. We estimate these side-suit tables' entries by scanning millions of games played by humans, considering only those tricks in which at least one side-suit card has been played by any player, and examining the outcome of those tricks. For example, if a player p holds the $\heartsuit KQ$ and plays the $\heartsuit Q$ in a trick consisting of $\heartsuit AQ7$, where an opponent plays the $\heartsuit A$, then we consider the $\heartsuit KQ$ configuration to have lost p 14 points (because the configuration allowed the opponent to both get an Ace home and capture the Queen), and to have lost both one trick and one high-card. The trump-related tables each have $2048 \cdot 4 \cdot 3 = 24576$ entries which are computed similarly, now focusing on tricks involving trump cards. The grand-game evaluation structure is also similar, except that up to 4 *trump targets* and *vulnerable* high cards are considered and the trump context is replaced by an index that encodes the Jack constellation and whether the soloist is in first playing position.

Bidding, Discarding, and Choosing Contracts

The 10+2 evaluation can be used for bidding as illustrated in Figure 4.2. The root value is the result of an expectimax computation of winning probability over all soloist choices (picking up the skat or not, selecting the game type (6 choices) to be played, and discarding two cards (66 choices)) as well as chance nodes for the initial skat (22 choose $2 = 231$ possibilities).

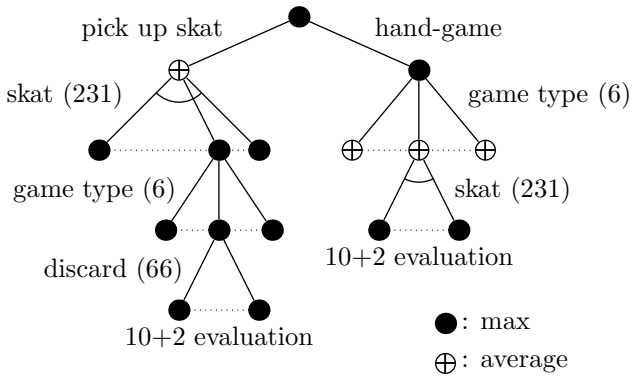


Figure 4.2: The simplified skat bidding search tree based on 10+2 evaluation.

Note that the hand-game branch of the tree is a convenient simplification in that we let the soloist "know" the skat cards so as to reuse the 10+2 evaluation function. This can cause the hand-game

evaluation to be overly optimistic.

In the bidding phase, our program evaluates up to 92,862 leaf nodes with the 10+2 evaluation function in a fraction of a second on modern hardware. The program then bids iff the root value meets or exceeds a fixed bidding threshold $B \in [0, 1]$. Lowering B leads to riskier bidding. We chose $B = 0.6$ from self-play experiments.

Once Kermit has won the bidding, the tree is searched again to determine whether or not to pick up the skat. If not, the hand-game with the highest winning probability is announced. Otherwise, Kermit receives the skat cards and selects the contract and the cards to discard based on searching the relevant subtree and maximizing the expected game value.

4.3 Inference

While skat has been studied by other authors, none of the previous work involves the inference of opponent cards during play. Research which uses inference in the closest manner to what we describe here includes work by Ginsberg in bridge [15], and Richards and Amir in Scrabble [32]. Ginsberg's bridge program, GIB, first places constraints on the length of card suits in each player's hand based on their bidding. For each such generated hand, GIB then checks whether it would have made the bid that was actually made in the game, throwing out hands that do not pass this test. Inference information from cardplay is incorporated as well, although precise details of how this is done are not presented in Ginsberg's work.

Richards and Amir's Scrabble program, Quackle, performs a simple form of inference to estimate the letters left on an opponent's rack (the *leave*) after the opponent has made a play. The probability $P(\text{leave} \mid \text{play})$ is estimated using Bayes' Rule. $P(\text{leave})$ is the prior probability of the letters left on the opponent's rack, obtained analytically from tile-counting. $P(\text{play})$ is a normalizing constant, and $P(\text{play} \mid \text{leave})$ is obtained by assuming the opponent is playing greedily according to Quackle's static move-evaluation function; that is, $P(\text{play} \mid \text{leave}) = 1$ if $play$ is the best-ranked play possible, given what letters the player is assumed to have had, and 0 otherwise. These probabilities are then used to bias the hypothetical worlds that Quackle examines during its simulations.

4.3.1 Inference Formulation

Our work has two key differences from that of Richards and Amir. The inference problem in imperfect information games can generally be described as the problem of finding $P(\text{world} \mid \text{move})$ for an arbitrary hypothetical world and some move made by another player. An agent capable of playing the game can obtain $P(\text{move} \mid \text{world})$, which easily leads to $P(\text{world} \mid \text{move})$ via Bayes' Rule. There are, however, two problems with this approach. The first, as Ginsberg and Richards and Amir point out, is that calculating $P(\text{move} \mid \text{world})$ for a large number of hypothetical worlds is intractable. The second is that if the computer player generates only deterministic moves (as is the

case with GIB, Quackle and indeed our own program), then $P(\text{move} \mid \text{world})$ will always be either 1 or 0. This makes the prediction brittle in the face of players who do not play identically to ourselves.

Therefore, the first key aspect of our work is that we obtain values for $P(\text{world} \mid \text{move})$ by training on offline data, rather than attempting to calculate $P(\text{move} \mid \text{world})$ at run-time. The second key aspect is that we generalize the inference formulation such that we can perform inference on high-level features of worlds, rather than on the individual worlds themselves. Examples of such features include the number of cards a player holds in a particular suit and how many high-point cards she holds. So long as we assume independence between features and conditional independence between worlds and moves given these features (which may not be true, but allows for tractability), we can derive a simple expression for $P(\text{world} \mid \text{move})$ as follows.

Here, let w be a world, m our move and $F = f_1 \dots f_n$ the features present in w. By definition of conditional probability, we have:

$$P(w|m) = \frac{P(w \wedge m)}{P(m)} \tag{4.1}$$

Because $P(F|w) = 1$, i.e. the presence of features is determined solely by the world, we can write:

$$P(w \wedge m) = P(w \wedge m \wedge F) \tag{4.2}$$

Now, using Bayes' Rule, consider the following:

$$P(w|F) = \frac{P(F|w)P(w)}{P(F)}, P(m|F) = \frac{P(F|m)P(m)}{P(F)} \tag{4.3}$$

Recall our assumption that worlds and moves are conditionally independent given features, i.e. $P(w \wedge m|F) = P(w|F)P(m|F)$. Therefore, combining this with equation 4.3, we can write:

$$P(w \wedge m|F) = \frac{P(F|w)P(w)P(F|m)P(m)}{P^2(F)} \tag{4.4}$$

By definition of conditional probability, $P(w \wedge m|F) = \frac{P(w \wedge m \wedge F)}{P(F)}$. Therefore, recalling that $P(F|w) = 1$, we have:

$$P(w \wedge m \wedge F) = \frac{P(w)P(F|m)P(m)}{P(F)} \tag{4.5}$$

Combining this with equations 4.2 and 4.1 and cancelling out $P(m)$, we get:

$$P(w|m) = \frac{P(w)P(F|m)}{P(F)} \tag{4.6}$$

Finally, recall that the features $F = f_1 \dots f_n$ were assumed to be independent. Therefore, we arrive at our final equation:

$$P(w \mid m) = P(w) \cdot \prod_i \frac{P(f_i \mid m)}{P(f_i)} \tag{4.7}$$

This is the formulation used by Kermit to perform inference. $P(f_i \mid m)$ in this case comes from a database look-up, while $P(f_i)$ adjusts for the number of worlds in which feature f_i is present, and can be calculated analytically at run-time. This inference can be easily extended to observing move sequences and used in card-playing programs to bias worlds generated in decision making modules.

4.3.2 Application to Skat Cardplay

The current version of our skat program draws conclusions from bids and the contract based on (4.7), assuming independence, and biases worlds generated for PIMC accordingly. We distinguish between soloist and defender card inference because of the inherent player asymmetry in skat.

As soloist, our program first samples possible worlds w_i uniformly without replacement from the set of worlds that are consistent with the game history in terms of void-suit information and soloist knowledge. Given $\text{bid}_i :=$ opponent i's bid, the program then computes

$$P(w_i \mid \text{bid}_1, \text{bid}_2) = P(h_1(w_i') \mid \text{bid}_1) \cdot P(h_2(w_i') \mid \text{bid}_2),$$

where w_i' represents the 32-card configuration in the bidding phase reconstructed from w_i by considering all played cards as well as the soloist's cards. Functions h_i extract the individual hands the opponents were bidding on from these complete 32-card configurations, and those are then evaluated with respect to the bid. In a final step, our program then samples worlds for PIMC using these values after normalization.

To estimate the probability of ten-card hands w.r.t. bids we could in principle make use of the bidding system we described in Subsection 4.2.2. However, we can only process 160 states per second, which is not fast enough for evaluating hundreds of thousands of possible hands in the beginning of the cardplay phase. Instead, we use table-based features to quickly approximate hand probabilities by multiplying feature value frequencies that have been estimated from generated bidding data. The following is a list of the features used.

- *Suit length* $\in \{0 \ldots 7\}$: The number of cards held in each of the four suits, ♢♡♠♣.
- *Jack constellaion* $\in \{0 \ldots 15\}$: The exact configuration of the player's Jacks.
- *Ace count* $\in \{0 \ldots 4\}$: Number of Aces held.
- *High card count* $\in \{0 \ldots 8\}$: Number of Aces and Tens held.
- *King Queen count* $\in \{0 \ldots 8\}$: Number of Kings and Queens held.
- *Low card count* $\in \{0 \ldots 12\}$: Number of 7s, 8s and 9s held.

The same basic algorithm is used for defenders modelling the soloist, but instead of basing the inference on the soloist's bid, we estimate the hand probability according to the announced contract, which is more informative than the bid. Because we are sampling the soloist's hand together with the two cards discarded in the skat, we can use our 10+2 evaluation directly, since there is no need for the tree search that averages over possible skats and game types. For this inference, we simply assign a zero probability to hands with a winning probability below a fixed threshold of 0.5. This is lower than Kermit's bidding threshold of 0.6 to account for the fact that Kermit's bidding is based on expected value, and it may find worse-than-average cards in the skat.

4.4 Kermit Versus the World: Experimental Results

In this section we present performance results for our skat program, *Kermit*, that indicate that it is much stronger than other existing programs and appears to have reached human expert strength. Measuring playing strength in skat is complicated by the fact that it is a 3-player game in which two colluders can easily win — as a team — against the third player. For instance, one colluder could bid high to win the auction and his "partner" lets him win by playing cards poorly. In this extreme form, collusion can be detected easily, but subtle "errors" are harder to spot. As this isn't a "feature" of our program, we score individual games using the Fabian-Seeger system which is commonly used in skat tournaments. In this system, soloists are rewarded with the game value plus 50 points if they win. Otherwise, soloists lose double the game value plus 50 points. In a soloist loss, each opponent is also rewarded 40 or 30 points, depending on whether 3 or 4 players play at a table.

4.4.1 Playing Local Tournaments

We first compare the playing strength of our program by playing two kinds of tournaments to differentiate between cardplaying and bidding strength. For this purpose we generated 800 random hands in which, according to our bidding system using threshold 0.6, at least one player makes a bid. In the cardplay tournaments we then had two competing players play each of those games twice: once as soloist and once as defenders, for a total of 1600 games. In each game our bidding system determined the soloist, the contract, and which cards to discard. In the full-game tournaments two programs compete starting with bidding and followed by cardplay. To take player position and defender configurations into account, we played 6 games for each hand (i.e., ABB, BAB, BBA, AAB, ABA, BAA for players A and B), totalling 4800 games. All tournaments were played on 2.5 GHz 8-core Intel Xeon 5420 computers with 6–16 GB of RAM running Linux/GNU. Our skat program is written in Java and compiled and run using Sun Microsystem's JDK 1.6.0.10. The other programs we tested are written in C/C++ and compiled using gcc 4.3. Each program was given 60 CPU seconds to finish a single game.

Table 4.2 lists the results of tournaments between various Kermit versions, XSkat, and KNNDDSS. XSkat is a free software skat program written by Gunter Gerhardt (`www.xskat.de`). It

Table 4.2: 2-way tournament results. S and D indicate that Kermit infers cards as soloist or defender, resp. The NI program version does not do any inference. In full-game tournaments Kermit uses bidding threshold 0.6.

Type	Name	Point Avg.	Std.Dev.	#
Cardplay	Kermit(SD)	**996**	50	1600
	Kermit(NI)	779	54	1600
Cardplay	Kermit(SD)	**986**	51	1600
	Kermit(S)	801	53	1600
Cardplay	Kermit(SD)	**861**	53	1600
	Kermit(D)	820	54	1600
Cardplay	Kermit(SD)	**1201**	48	1600
	XSkat	519	56	1600
Cardplay	Kermit(SD)	**1012**	51	1600
	KNNDDSS	710	53	1600
Full Game	Kermit(SD)	**1188**	30	4800
	XSkat	629	26	4800
Full Game	Kermit(NI)	**1225**	30	4800
	XSkat	674	27	4800
Full Game	Kermit(SD)	**1031**	32	4800
	KNNDDSS	501	21	4800

is essentially a rule-based system which does not perform search at all and serves as a baseline. KNNDDSS is described by Kupferschmid *et al.* [23, 19]. It uses k-nearest-neighbor classification for bidding and discarding and PIMC search with uniformly random world selection for cardplay.

Listed are the point averages per 36 games (which is the usual length of series played in human tournaments), its standard deviation, and the total number of games played. Averaging 950 or more points per series when playing in a skat club is regarded as a strong performance among human players. As indicated in the top three rows, adding card inference to Kermit makes it significantly stronger, with defenders inferring the soloist's cards having the biggest impact. The following two rows show that the cardplay of XSkat and KNNDDSS, which do not use card inference apart from tracking void suits, is weaker than Kermit(SD)'s. In the full game tournaments XSkat's performance is slightly better, which means that its bidding performance is better than its cardplay relative to Kermit. The similar performance between the inference and no-inference versions of Kermit against XSkat indicate that Kermit's inference is robust enough that its quality of play is not compromised even against an opposing player very different from itself. The performance drop of KNNDDSS when playing entire games is caused by Kermit outbidding it by a rate of more than 3:1, even at an aggressive bidding threshold of 51 points used for KNNDDSS. It is possible that this is due to a bug in KNNDDSS, or it may be an artefact of the issues we discussed in Section 4.2, where open-handed and PIMC search based evaluations (which were used to train the knn classification of KNNDDSS) severely underestimate the soloist's winning chances.

4.4.2 The International Skat Server (ISS)

In late 2007, we launched the International Skat Server (ISS), an online environment which allows human and computer players to compete at skat. ISS is written in Java, and a platform-independent graphical user interface is available from the ISS website (`www.skatgame.net/iss`) which players can download and use to connect to our server.

Play on ISS is organized around virtual tables. When a human player wishes to start a game, he creates an empty table and then invites two or three other players to join that table, at which point the assembled group of players will most likely play several hands together. Several computer players — including Kermit — are "idling" on ISS at all times, and will accept any invitation that they receive. The computer players are capable of playing at multiple tables simultaneously, or in multiple instances at the same table. The computer players do not initiate game invitations on their own. This format favours the human players, since they are free to explicitly seek out weak opponents or even to always play with the same friendly human "partner" in all of their games.

For the play of a single hand, all players on ISS are allotted a maximum of three minutes play-time. If a player's clock runs out, he (and his partner if he is a defender during cardplay) forfeits the match. The computer players on the server typically do not come close to using their entire allotted time. The ISS user interface also allows human players an option to play in a (fascetiously named) "noob mode," in which they may examine the trick history and view the cards that have not yet been played during the game. At a face-to-face skat table, players are not allowed to examine this information, but we provide this option on the server to eliminate the computer players' advantage of a perfect memory (if the human players so desire).

To provide a rough picture as to the popular useage of ISS, as of June 2011, there are 126 human players who have played 200 games or more on the server; the most active human player has played over 49,000 games and 20 of the human players have played over 10,000 games.

4.4.3 Kermit's Online Play

Since its creation in 2008, Kermit has been playing on ISS. At the time when we first published results outlining Kermit's playing strength in 2009, we presented the server results shown in Table 4.3. The top entries list the results of Zoot, Kermit, XSkat, and Bernie, the four computer players playing on the server, playing against Zoot and Kermit. Zoot is a clone of Kermit using bidding threshold 0.55 instead of 0.6, meaning that it bids more aggressively and is willing to play riskier-looking games. Bernie is a program written by Jan Schäfer [38] that uses UCT search with XSkat as playout module in all game phases, as mentioned in Chapter 3, Section 3.2.2. These results show that the Kermit variations dominate these two programs when averaging over all games in which the programs happened to play together at the same virtual table against other players on the server. The bottom entries presents the results for Ron Link and Eric Luz, who are members of the Canadian team that won the skat world-championships in 2006 and 2008. Although some entries do not

Table 4.3: Skat server game results. colPts and rowPts values indicate the point average for the column and the row players, resp, while the # column is the number of hands played. Standard deviations are shown in parentheses.

Name	Zoot			Kermit		
	rowPts	colPts	#	rowPts	colPts	#
Zoot	**942** (25)	942 (25)	6.4k	876 (43)	**888** (44)	2.3k
Kermit	**888** (44)	876 (43)	2.3k	919 (11)	**919** (11)	36k
XSkat	577 (49)	**1116** (44)	1.9k	592 (46)	**1209** (43)	2.0k
Bernie	407(185)	**1148**(177)	133	588 (66)	**1126** (65)	998

Name	Ron Link			Eric Luz		
	rowPts	colPts	#	rowPts	colPts	#
Zoot	**927** (141)	604 (137)	213	**918** (38)	860 (38)	2.7k
Kermit	876 (101)	**898** (99)	408	**836** (43)	739 (42)	2.3k
XSkat	572 (75)	**1075** (72)	812	**829** (296)	760 (300)	50
Bernie	553 (205)	**987** (192)	108	**833** (135)	370 (146)	217

indicate a statistically significant advantage for either side, it is apparent that Kermit is competing at expert-level strength.

As of June, 2011, Kermit had played over 354,000 games on ISS, with a list average of 937, which suggests that even after several years of opportunity to probe for weaknesses, the human players on the server have been unable to find a way to significantly exploit Kermit's play.

Further evidence of Kermit's considerable playing strength was provided by Dr. Rainer Gößl, a strong skat player from Germany, who maintains a skat website that contains up-to-date reviews of commercial skat programs (`www.skatfox.com`). After playing at least 900 games against each program he concludes by saying that today's best programs stand no chance against good players. The best program he tested achieved an average of 767 points against his own 1238. When playing 49 games with Kermit, Bernie, and XSkat he finished with a 751 point average, while the programs achieved 1218, 1007, and -34 points, respectively. While not being statistically significant by itself, this outcome in combination with the other results we have presented and Gößl's program reviews suggests that at the moment Kermit is stronger than any other program and plays skat at human expert level.

4.5 Why Kermit?

The curious reader may wonder why we chose the name "Kermit" for our program. It is, of course, a reference to Jim Henson's legendary muppet Kermit the Frog, as featured prominently on the excellent television programs *The Muppet Show* and *Sesame Street*. When we launched ISS in 2007, the first computer player to "live" on the server was the UCT player by Jan Schäfer. At that time, the ISS infrastructure could not handle two instances of the same login name playing at the same virtual table. Therefore, it was necessary to run two entirely separate instances of the UCT player,

each with its own handle; since these players came as a pair, we decided to use the logins Ernie and Bert, after the inseparable muppet room-mates from *Sesame Street*. From there, it seemed natural that our next program should also be named for a well-known muppet and so Kermit emerged as the clear favourite. As the ISS infrastructure improved, Ernie and Bert were merged into the single user name of Bernie, an obscure reference to their once-separate identities.

4.6 Conclusion

In this chapter, we have presented our skat-playing program, Kermit, which has been shown to substantially defeat existing skat programs and plays roughly at the level of human experts. We have also quantified the performance gain that Kermit achieves from its inference abilities, which has not generally been done in other card-playing programs. Finally, Kermit owes its performance in bidding to a state evaluator built from human data that drastically outperforms other existing bidding methods.

For the remainder of this dissertation, Kermit is generally the standard against which additional techniques and refinements are tested. Whenever we refer to Kermit without qualification, we are referring to precisely the player we have described here.

When you can't beat 'em, join 'em.
-English Proverb

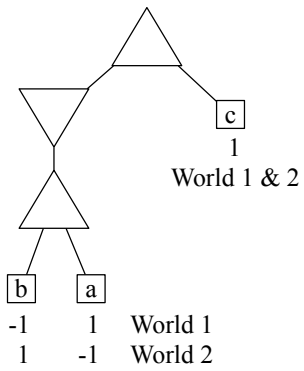
5 Why Perfect Information Monte Carlo Search?

Although it has been used to create strong computer players in the past, Perfect Information Monte Carlo (PIMC) search is a technique with known theoretical failings. When we first began work to create a strong computer skat program, one of our initial goals was to find a technique that could overcome these failings and surpass PIMC search as an algorithm for cardplay in games with a similar structure to skat. We had hopes, for example, of employing the UCT algorithm that had revolutionized the world of computer Go. However, the performance of Kermit, using a relatively straight-forward PIMC implementation as described in Chapter 4, and its convincing domination of other computer players, including one using a UCT approach, caused us to reconsider this goal. We devote this chapter to a deeper understanding of the strengths and weaknesses of PIMC search, and to why it works as well as it does in the domains in which it has been applied.

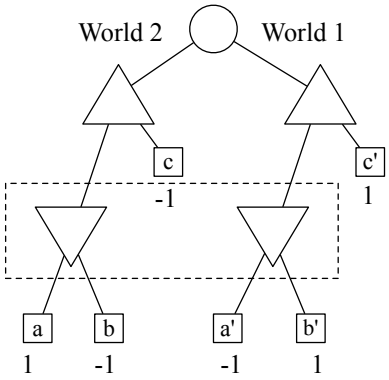
5.1 Understanding the Failings of PIMC Search

It is well-known that PIMC search as an algorithm does not produce a Nash equilibrium or any other strategy with provable game theoretic properties. Frank and Basin were the first to precisely identify and formalize the exact nature of the errors that PIMC search will make [9]. Fundamentally, these errors occur because repeated plays of the perfect information variant of a game do not truly capture all of the imperfect information aspects of the original game.

Frank and Basin distinguish between two distinct types of errors that PIMC search is guaranteed to be unable to detect, irrespective of the number of hypothetical worlds examined. The first of these errors is termed *strategy fusion*. Strategy fusion arises because Perfect Information Monte Carlo search (incorrectly) believes it has the luxury to choose the best strategy for any particular world, whereas in reality, there are situations (or information sets) which consist of multiple different perfect information scenarios. In the full imperfect information game, a player cannot distinguish between these scenarios, and must choose the same strategy in each one; but PIMC search erroneously assumes that it can choose a strategy tailored to each individual scenario. We illustrate strategy fusion in Figure 5.1(a), which depicts an abstract, two-player zero-sum game tree. The maximizing player is represented as an upward pointing triangle, and the minimizing player by a downward triangle. Terminal nodes are squares with payoffs for the max player below them. There are two worlds which would be created by a chance node higher in the tree. We assume neither



(a) Strategy Fusion



(b) Non-locality

Figure 5.1: Examples of strategy fusion and non-locality.

player knows whether they are in world 1 or 2, so we do not show this information in the tree.

At the root, the maximizing player has the choice of moving to the right to node (c) where a payoff of 1 is guaranteed, no matter the world. The maximizing player can also get a payoff of 1 from the node marked (a) in World 1 and the node marked (b) in World 2. PIMC search will think that it can always make the right decision above nodes (a) and (b), and so both moves at the root look like wins. However, in reality the max player is confused between worlds 1 and 2 and may actually make a mistake in disambiguation on the left side of the tree. We note that there are two conditions required for strategy fusion to actually cause an error in the play of PIMC search. First, there must be moves which are anti-correlated values (nodes (a) and (b)) on one portion of the tree, and second, there must be a move which is guaranteed to be better on the other side of the tree. If node (c) had the value -1, PIMC search would make the correct decision, although it would overestimate the value of the tree.

The second error identified by Frank and Basin is termed *non-locality*. Non-locality is a result of the fact that in a perfect information game, the value of a game tree node is a function only of its subtree, and therefore the value of a node is completely determined by a search starting with

its children. In an imperfect information game, a node's value may depend on other regions of the game tree not contained within its subtree, primarily due to the opponent's ability to direct the play towards regions of the tree that he knows (or at least guesses) are favorable for him, using private information that he possesses but we do not. This phenomenon creates non-local dependencies between potentially distant nodes in the tree.

We illustrate non-locality in Figure 5.1(b). In this figure there is a chance node at the top of the tree. The maximizing player knows the chance action, but the minimizing player cannot distinguish between the states within the dotted rectangle. In this tree PIMC search would conclude that all moves for the minimizing player are of equal value, and would therefore select a move at random. But, in fact, the minimizing player can always know the correct move. Because the maximizing player will take the win in world 1 if possible, the minimizing player will only have an opportunity to play if he is in world 2, when the maximizing player moves to the left to avoid the immediate loss. Thus, the minimizing player can infer the correct world and the correct action.

In Chapter 3, we have previously mentioned Frank and Basin's *Best Defense* model, which aspired to reconcile computer analysis of bridge hands with that of human experts. The effective result of this model is that the strategy fusion problem is removed from the point of view of the maximizer; the minimizer is assumed to still have perfect information. Ginsberg's lattice search achieves a similar effect, and is primarily concerned with ensuring the maximizer (particularly the declarer in the context of bridge) can distinguish "clever" lines of play that win no matter the split of the cards between the defenders from lines that require a deferred guess on the maximizer's part. However, neither of these techniques address an issue that frequently arises in skat, which is the maximizer's need to recognize and exploit the ignorance of his opponents.

To better illustrate this problem, we consider the more concrete skat-like example presented in Figure 5.2. Say that there are no trump in this game, and that forehand, who holds the $\diamondsuit A$, the $\clubsuit 10$ and the $\spadesuit 10$, is the soloist; she gets to discard one of her three cards at the start of the game. Let us also say she needs to win both tricks to win the game. Clearly, forehand's best choice is to keep the $\diamondsuit A$ and discard one of the Tens. To a human player, it is also clear how to play the hand: forehand should begin with the $\diamondsuit A$, at which point middlehand (who holds the $\clubsuit A$ and the $\spadesuit A$) must drop one of his two aces. However, as he does not know which card forehand discarded, he does not know which ace to keep. All else being equal, there is a 50% chance then that he drops the wrong ace. If he does so, then forehand will play her remaining Ten and win the second trick as well. Thus, with this strategy, forehand is guaranteed one trick and has a 50% chance of making two and winning the game.

However, this is not the conclusion that Perfect Information Monte Carlo search will come to. Because PIMC search analyzes the perfect information version of the game, the defenders will always *know* which of the Tens forehand discarded, and which one she kept. Therefore, in the perfect information play-outs, middlehand will *always* keep the correct ace, and therefore prevent

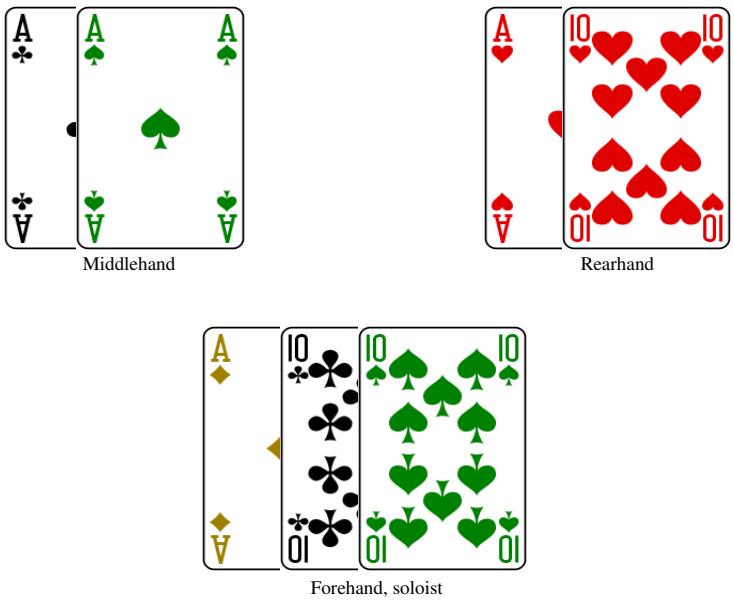


Figure 5.2: A cardplay situation that is incorrectly analyzed by PIMC search.

forehand from taking her second trick. What's worse, since forehand needs both tricks to win the game, when considering her first move, she will (incorrectly) see that all moves lead to a loss, and is therefore equally likely to lead the $\diamondsuit A$ as she is her Ten, when of course in reality, leading the Ten guarantees her loss, while leading the ace means a 50% shot at victory. This is indeed an example of strategy fusion, but neither Frank and Basin's Best Defense nor Ginsberg's lattice search could resolve this scenario, since it requires recognizing that the opponents do not in fact have perfect information and using this fact to make a potentially game-winning gambit.

In the example in Figure 5.2, we can perhaps save PIMC search by writing our alpha-beta search such that it prefers maximizing card points won even if an overall win is impossible, which is the approach taken by our program Kermit as described in Chapter 4. However, in Figure 5.3, we present a situation in which this modification to alpha-beta not only fails, but in fact specifically hurts our situation. This time, there is still no trump and middlehand is the soloist, holding $\clubsuit A$, $\spadesuit 8$ and $\spadesuit 7$. Forehand holds the $\clubsuit 10$ and the $\heartsuit 10$, while his partner holds the $\heartsuit A$ and the $\diamondsuit A$. The decision of interest here is which card the soloist will discard. If none of the players know each others' cards, then forehand has a dilemma. He needs to play the Ten in the suit in which his partner holds the ace, but he does not know which suit this is. Clearly then, if middlehand keeps the $\clubsuit A$, there is a chance that forehand plays the $\clubsuit 10$, allowing middlehand to take the first trick, and then take the second trick in spades since the defenders hold none. Of course, discarding the $\clubsuit A$ guarantees the soloist's loss.

However, again in the perfect information analysis of this deal, forehand will never make the mistake of playing the wrong Ten. Therefore, as far as middlehand can tell, all moves are a guaran-

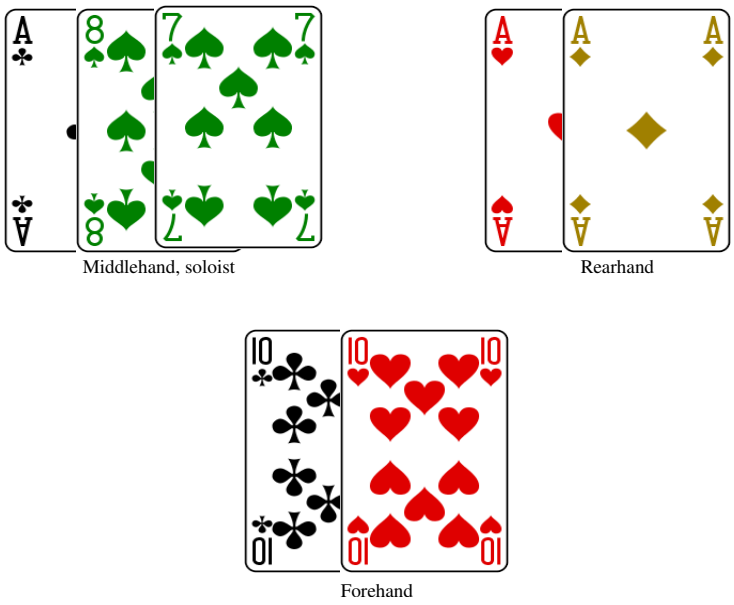


Figure 5.3: A cardplay situation in which optimizing card points leads to the worst possible performance.

teed loss. However, if our alpha-beta search has been tweaked to prefer moves that maximize card points, then middlehand will in fact *prefer* the situation where he discards the $\clubsuit A$, because at least then that card is safe — even though in so doing, he gives up his 50% chance of taking both tricks and therefore winning the game!

In this second situation, the analysis above only shows that alpha-beta search incorrectly analyzes a single hypothetical world (though in this case, the true one). The PIMC search that averages over all worlds would indeed find in this situation that keeping the $\clubsuit A$ is the right move for middlehand, since there are card permutations — for instance, swapping the $\heartsuit A$ and $\heartsuit 10$ — in which keeping $\clubsuit A$ will win a trick. However, let us assume that somehow, middlehand learns more about the game, and infers the cards that his opponents hold, without them being able to do the same. In this case, middlehand now knows the true world, and since alpha-beta incorrectly analyzes this world, middlehand is now guaranteed to make the worst possible move. Thus, we have created the perverse situation where knowing more actually *hurts* the performance of our player!

5.2 Understanding the Success of PIMC Search

The discussion of the preceding section may give the impression that the situation for PIMC search is quite bleak. Indeed, PIMC search has often been criticized for "avoiding the issue" of imperfect information. For instance, in the 2nd edition of their widely-used AI textbook, Russell and Norvig [34] write that PIMC search (which they call "averaging over clairvoyancy") suggests a course of action that "no sane person would follow" in a simple example that they present (in the 3rd edition, the

language is softened, although the same example is shown [35]).

Given such criticism, the true mystery is not why PIMC search *might* not work, but rather why it works as well as it does, at least in the domains we have discussed. If PIMC search was fundamentally the wrong approach, one would expect that no program of reasonable quality could use it to play at any level approaching human strength. In this chapter, we will attempt to answer why PIMC search has in fact been quite successful in practice.

We hypothesize that there should be a small number of concise properties which can describe the general properties of a game tree and measure, from these properties, whether PIMC search as an approach is likely to be successful or not. Several properties are proposed and used to build small synthetic trees which can be solved and also played with PIMC search. We will show that these properties directly influence the performance of PIMC search, and that they can be measured in real games as well. Thus, we are able to suggest that in many classes of games PIMC search will not suffer large losses in comparison to a game-theoretic solution.

The author expresses his thanks to Dr. Nathan Sturtevant and Tim Furtak for their contribution to the work that follows. In particular, Dr. Sturtevant provided the data and analysis relating to hearts and Kuhn poker, and Tim is responsible for measuring our synthetic tree properties in real games.

5.2.1 Motivation

In spite of its theoretical drawbacks, there is evidence that PIMC search can produce strong computer players in some domains, in particular trick-taking card games such as bridge and skat. We have seen in Chapter 4 how in skat, it dominates computer players based on hard-coded rules and UCT, and even performs well against expert humans. Perhaps, however, the "bar is low," as it were, in these domains; even good human players may not actually play these games well in an absolute sense. How well would PIMC search perform against a game-theoretically optimal player?

As we have previously discussed, finding a Nash equilibrium to the full game of skat is infeasible. However, it *is* possible to "solve" (with some caveats) small skat end-game positions, with for instance three cards remaining for each player. We will do this using the CFR algorithm that has been used to compute equilibrium strategies for poker [18], and compare the resulting optimal solution against a PIMC search player. For solving purposes, we will consider this three-trick sub-game to begin with a chance node that distributes the remaining 11 cards among the three players and the skat in a manner that is consistent with known void-suit information but is otherwise uniform. This uniform distribution is, of course, not precisely accurate in describing the true game state, but we will use the same assumption for both our equilibrium player and for PIMC search which will hopefully make for a fair comparison. For our starting positions, we will use real skat games played by humans for the first seven tricks for which the outcome of the game is still uncertain (i.e. neither side has yet acquired sufficient card points to win the game). Empirically, approximately 15% of skat games are still unresolved when there are three tricks left in the game, based on our analysis of

thousands of human-played games.

Over a randomly selected set of 3,000 such unresolved games, PIMC search makes mistakes that cost it 0.42 tournament points (TPs) per deal on average against the equilibrium solution computed by CFR. We must also note the caveat that CFR is not guaranteed to produce Nash equilibrium solutions to multi-player games such as Skat; however, there is evidence that it still often produces strong strategies [33], especially for small games of the type considered here. If we assume the value of 0.42 to be close to the true loss against a Nash-optimal player, then as only 15% of games are unresolved at this point, PIMC's average loss is only 0.063 TP per deal. In a series of 180 deals for each player (5 lists of 36 games each) in typical Skat tournaments the expected loss amounts to 11.3 TPs, which is dwarfed by the empirical TP standard deviation of 778. Thus, the advantage over PIMC search in the endgame hardly matters for winning tournaments. If we assume that this result generalizes to the ten-trick game — a big assumption, to be sure — a loss of 0.42 TPs per deal, or 15 TPs per list, is more significant, but still quite modest. It seems enough, in any case, to warrant further investigation into the question of how much PIMC search will suffer against an equilibrium player in different domains.

5.3 Methodology

As we have described in the preceding sections, work by Frank and Basin has already formalized the kinds of errors made by PIMC search through the concepts of strategy fusion and non-locality. However, not only does the mere presence of these properties seem difficult to detect in real game trees where computing a full game-theoretic solution is infeasible, but as we have previously argued, their presence alone is not enough to necessarily cause PIMC search to make mistakes in its move selection.

Therefore, instead of focusing on the concepts of strategy fusion and non-locality directly, our approach is to measure elementary game tree properties that probabilistically give rise to strategy fusion and non-locality in a way that causes problems for PIMC search. In particular, we consider three basic properties.

- *Leaf Correlation*, lc, gives the probability all sibling terminal nodes have the same payoff value. Low leaf node correlation indicates a game where it is nearly always possible for a player to affect their payoff even very late in a game.

- *Bias*, b, determines the probability that the game will favor a particular player over the other. With very high or very low bias, we expect there to be large, homogeneous sections of the game, and as long as a game-playing algorithm can find these large regions, it should perform well in such games.

- *Disambiguation factor*, df, determines how quickly the number of nodes in a player's information set shrinks with regard to the depth of the tree. For instance, in trick-taking card games,

each play reveals a card, which means the number of states in each information set shrinks drastically as the game goes on. Conversely, in a game like poker, almost no private information is directly revealed until the game is over. We can determine this factor by considering how much a player's information set shrinks each time the player is to move.

All of these properties can be measured in real game trees, as we describe below.

5.3.1 Measuring Properties in Real Games

To measure leaf correlation, bias, and the disambiguation factor in real games (i.e. *large* games) we suggest using random playouts to sample the terminal nodes (unless there is some a priori reason to discard portions of the game tree as uninteresting or irrelevant). Once a terminal node is reached, the bias and correlation may be estimated from the local neighbourhood of nodes. Along these random playout paths the size of the information set to which each node belongs may be computed in a straightforward manner, and compared to the subsequent size when next that player is to move. It is then straightforward to convert the average reduction ratio into a df value for one's desired model.

5.4 Experimental Setup

In this section, we first describe experiments on synthetic trees measuring the performance of PIMC search in the face of various tree properties, and then show the measurement of these properties in the real games that make up our domains of interest.

5.4.1 Synthetic Trees

We construct simple synthetic trees used in our experiments as follows. We assume, for simplicity's sake, that our trees represent a two-player, zero-sum, stochastic imperfect information game. We may also assume, without loss of generality, that the game has alternating moves between the two players, and that all chance events that occur during the game are encapsulated by a single large chance node at the root of the game tree. Each player node is of degree 2, while the degree of the beginning chance node is defined in terms of worlds per player, W. Furthermore, the information concerning these worlds is assumed to be strictly disjoint; for each world of player p_1, there are

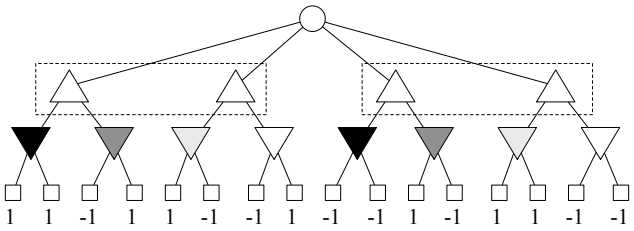


Figure 5.4: A sample of a depth 2 synthetic tree, with 2 worlds per player. Max nodes in boxes and min nodes with the same shading are in the same information set respectively.

initially W worlds for player p_2 that p_1 cannot distinguish. We restrict ourselves to this disjoint case because in cases where the players' information overlaps, the game collapses to a perfect information stochastic game (i.e. there may be information unknown to both players, but at least they are in the same boat). Therefore, the total degree of the chance node is W^2. Game tree nodes are initially partitioned into information sets based on these worlds. We assume that all player moves are observed by both players, in the sense that both players know whether the player to move chose the 'left' or 'right' branch at each of their information sets. Finally, terminal payoffs are restricted to be either 1 (a win for p_1) or -1 (a win for p_2). A small sample of such a tree is presented in Figure 5.4.

Now, under the above assumptions, we define our three properties in the synthetic tree context, each one of which is continuous valued in the range $[0, 1]$. We describe below the effect of these parameters on the construction of the synthetic trees.

- *Leaf Correlation*, lc: With probability lc, each sibling pair of terminal nodes will have the same payoff value (whether it be 1 or -1). With probability $(1 - lc)$, each sibling pair will be *anti-correlated*, with one randomly determined leaf having value 1 and its sibling being assigned value -1.

- *Bias*, b: At each *correlated* pair of leaf nodes, the nodes' values will be set to 1 with probability b and -1 otherwise. Thus, with bias of 1, all correlated pairs will have a value of 1, and with bias of 0.5, all correlated pairs will be either 1 or -1 at uniform random (and thus biased towards neither player). Note that *anti-correlated* leaf node pairs are unaffected by bias.

- *Disambiguation factor*, df: Initially, each information set for each player will contain W game nodes. Each time p is to move, we recursively break each of his information sets in half with probability df (thus, each set is broken in two with probability df; and if a break occurs, each resulting set is also broken with probability df and so on). If df is 0, then p never gains any direct knowledge of his opponent's private information. If df is 1, the game collapses to a perfect information game, because all information sets are broken into sets of size one immediately. Note that this generative model for df is slightly different than when measuring disambiguation in real games trees.

Note that we do not specifically examine correlation within an information set; rather, we hypothesize that these properties represent the lowest level causes of tree structures that result in problems for PIMC search, and that if they are present with sufficient frequency, then higher-level confusion at the information set level will occur.

5.4.2 Experiments on Synthetic Game Trees

Using the synthetic game tree model outlined above, we performed a series of experiments comparing the playing strength of both PIMC search and a uniform random player against an optimal

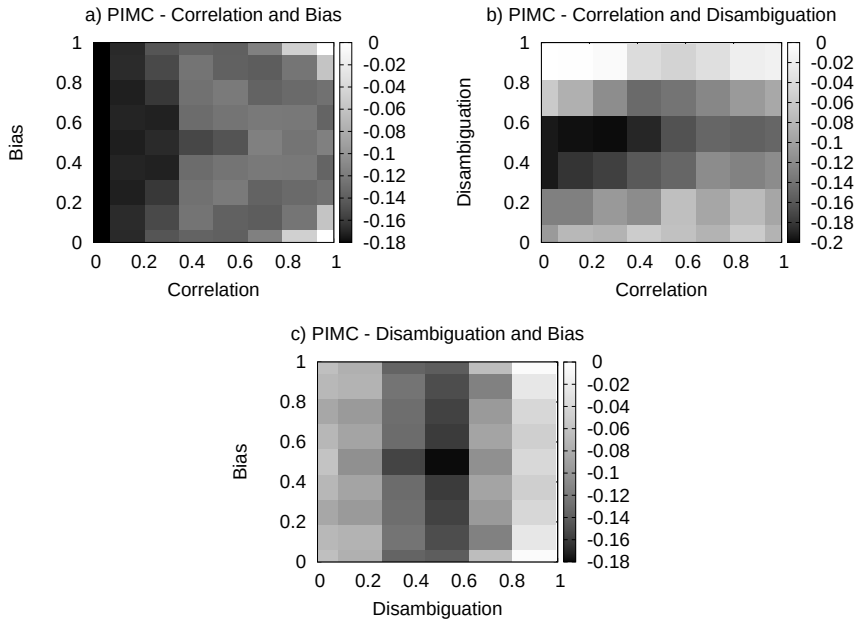


Figure 5.5: Performance of PIMC search against a Nash equilibrium. Darker regions indicate a greater average loss for PIMC. Disambiguation is fixed at 0.3, bias at 0.75 and correlation at 0.5 in figures a, b and c respectively.

Nash equilibrium player created using the CFR algorithm. In each experiment, synthetic trees were created by varying the parameters for leaf correlation, bias and disambiguation. Tree depth was held constant at depth 8, and we used 8 worlds per player at the opening chance node, for a total chance node size of 64. Playing strength is measured in terms of average score per game, assuming 1 point for a win and -1 for a loss. For each triple of parameter values, we generated 10000 synthetic trees and played 2 games per tree, with the competing players swapping sides in the second game.

The results of these tests are presented in figures 5.5 through 5.7. For ease of visualization, each figure plots two parameters against each other on the x and y axes, while the third parameter is held constant. Figures 5.5 and 5.6 shows the playing performance of the challenging player (either PIMC search or uniform random) against the equilibrium player. White shaded regions are areas of the parameter space where the challenger breaks even with equilibrium. The darker the shading, the greater the challenger's loss against the equilibrium. Figure 5.7 is similar, except that the shading represents the *gain* of PIMC search over the random player when playing against equilibrium. Dark regions of these plots represent areas where PIMC search is performing almost no better than the random player, whereas lighter regions indicate a substantial performance gain for PIMC search over random.

These plots show that PIMC search is at its worst when leaf node correlation is low. This is true both in absolute performance, and in PIMC's relative improvement over random play. The most likely explanation for this behavior is that when anti-correlation occurs deep in the game tree – particularly at the leaves – then PIMC search always believes that the critical decisions are going

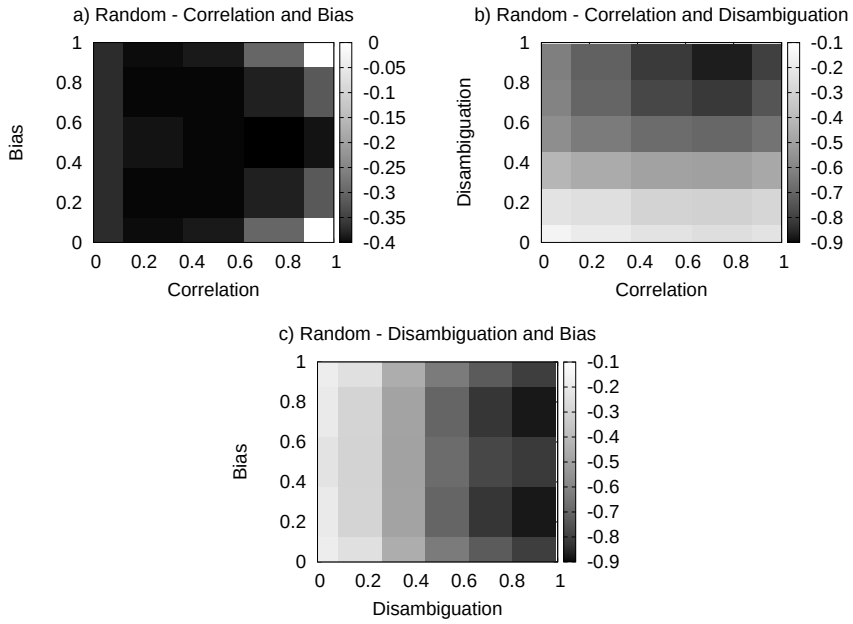


Figure 5.6: Performance of random play against a Nash equilibrium. Darker regions indicate a greater average loss for random play. Disambiguation is fixed at 0.3, bias at 0.75 and correlation at 0.5 in figures a, b and c respectively.

to come 'later' and that what it does higher up the tree does not actually matter. Of course, when an information set structure (which PIMC ignores at every node except the root of its own search) is imposed on the tree, early moves frequently *do* matter, and thus the superior play of the equilibrium player. When correlation is medium to low, bias also seems to play a role here, with more extreme bias resulting in better performance for PIMC, although the effect of bias is generally small. The performance gain due to bias for PIMC is likely because a more extreme bias reduces the probability of interior nodes that are effectively anti-correlated occuring perhaps one or two levels of depth up from the leaves of the tree. Note that, of course, in the case of maximum bias and correlation, even the random player will play perfectly, since the same player is guaranteed to win no matter what the line of play (we can only suppose these would be very boring games in real life).

The situation with the disambiguation factor initially appears counter-intuitive; it appears that a low disambiguation factor is good for the absolute performance of PIMC search, while the worst case is a mid-range disambiguation factor. However, in the relative case of PIMC's gain over random, the trend is very clearly reversed. The explanation for this lies in the fact that the random player performs relatively well in games with a low disambiguation factor. In some sense, because there is so much uncertainty in these games, there is a lot of 'luck,' and there is only so much an optimal player can do to improve his position. As we increase the disambiguation factor, the performance of the random player deteriorates rapidly, while PIMC search is much more successful at holding its own against the optimal player. As disambiguation approaches 1, the performance of PIMC improves drastically, since the game is approaching a perfect information game. Finally, we note

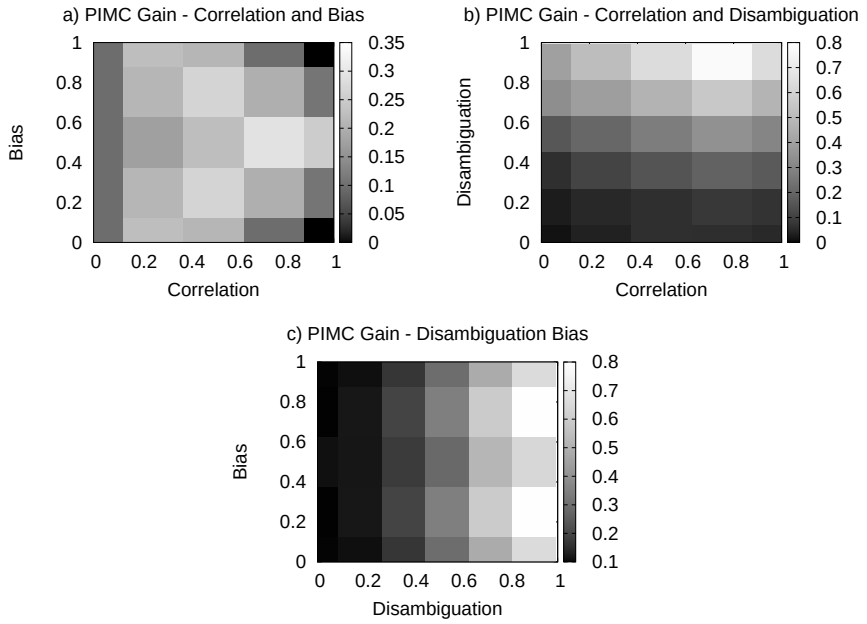


Figure 5.7: Performance gain of PIMC search over random against a Nash equilibrium. Darker regions indicate minimal performance gain for using PIMC search over random play. Disambiguation is fixed at 0.3, bias at 0.5 and correlation at 0.75 in figures a, b and c respectively.

that with a high disambiguation in the 0.7-0.9 range, low correlation is actually good for PIMC's performance. This is a result of the fact that these games become perfect information games very quickly, and low correlation increases the probability that a win is still available by the time the PIMC player begins playing optimally in the perfect information section of the tree.

5.4.3 Real Games

To test the predictive powers of our three properties, we estimated the distribution of those parameters for actual games. The first game so measured is skat. We differentiate results by the three major game types — suit, grand and null. For each game type, 10000 human-bid games were explored using random actions from the start of the cardplay phase. In each game correlation and bias were measured 1000 times near the leaves. To do this, we walk down the tree, avoiding moves which lead to terminal positions (after collapsing chains of only one legal move). When all moves lead directly to terminal positions we take their value to be the game value with respect to the soloist (to emulate the values of the fixed depth synthetic trees). We say these "pre-terminal" nodes are correlated if all move values are the same, and compute bias as the fraction of correlated nodes which are soloist wins.

Disambiguation was measured by comparing the change in the number of possible (consistent) worlds since the current player was last to move. Only 10 disambiguation rollouts were performed per world, since the resulting ratios were tightly clustered around $df = 0.6$. The observed distributions are shown in Fig. 5.8. In this figure, we also display results for Hearts, which, like skat, is a

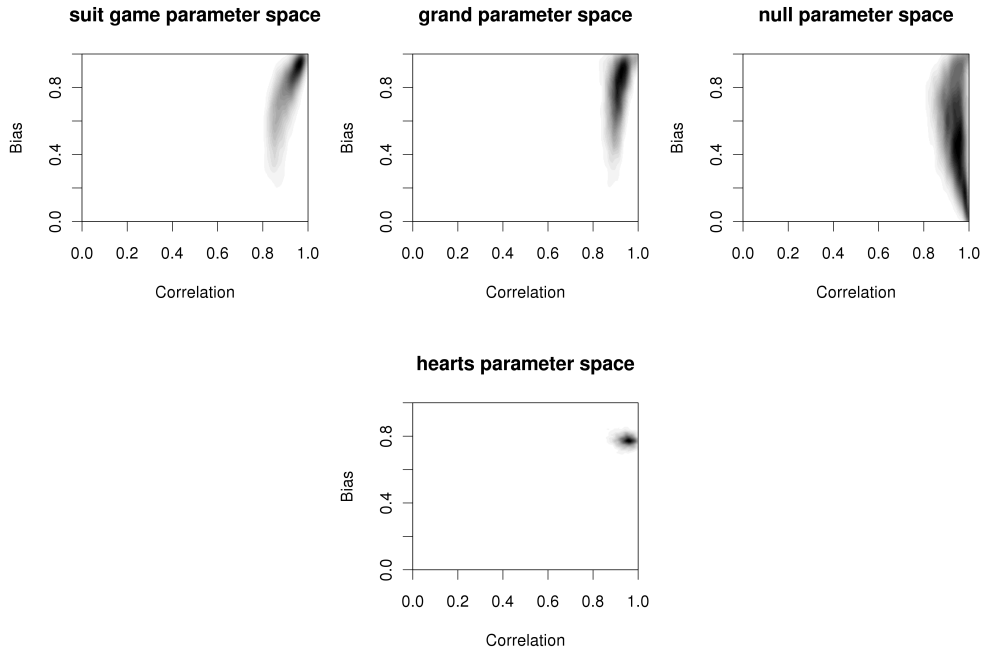


Figure 5.8: Parameter space estimation for Skat game types and Hearts. Dark regions correspond to a high density of games with those measured parameters. Values were sampled using 10000 games for each skat type and 3000 games for hearts. Bias is given in terms of score w.r.t. a fixed player.

trick-taking card game, but played with a larger deck and different scoring rules. 3000 Hearts games using 500 sample points per game were used to generate this data.

For both the skat and hearts games, the resulting graphs show a very high level of correlation (from 0.8 to nearly 1.0), with bias varying more widely and disambiguation very close to 0.6, as mentioned above. Examining Figures 5.5(b) and 5.7(b) puts skat in a parameter space where the PIMC player loses only 0.1 points per game against equilibrium and gains 0.4 points over random play (recalling that our synthetic trees use payoffs from -1 to 1), with perhaps plus or minus 0.05 points depending on the bias of the individual hand. This seems like relatively good performance for PIMC search, which coincides with our motivating evidence that PIMC search seems to perform well in these games in practice.

Another game we examined is Kuhn poker, a highly simplified poker variant for which Nash-optimal solutions are known. In this game two players are each dealt one card out of a deck of three. The game proceeds as: both players ante; player 1 may check or raise; player 2 may fold, check, call, or raise as appropriate; if the game is still proceeding, player 1 may fold or call. The player with the high card then wins the pot. With Nash-optimal strategies player 1 is expected to lose 1/18 = $0.0\bar{5}$ bets per game and player 2 to win 1/18 bets per game.

This game has a disambiguation factor of 0, since no cards are revealed (if at all) until the end, and the size of an information set is never decreased. By inspecting the game tree and using our

Table 5.1: Average payoff achieved by random and PIMC against Nash and best-response players in Kuhn poker.

	Opponent	
Player:	Nash	Best-Response
Random (p_1)	-0.161	-0.417
Random (p_2)	-0.130	-0.500
PIMC (p_1)	-0.056	-0.083
PIMC (p_2)	0.056	-0.166

notion of pre-terminal nodes the correlation and bias can be seen to be 0.5 and 0.5 respectively. These parameters are very different from skat and hearts and lie in the portion of parameter space where we would predict that PIMC search performs more poorly and offers little improvement over random play. This is then, perhaps, at least one explanation of why research in the full game of poker has taken the direction of finding game-theoretic solutions to abstract versions of the game rather than tackling the game directly with PIMC search.

We present results comparing play between a random player, PIMC player and Nash player in Table 5.1. Kuhn poker is not symmetric, so we distinguish the payoffs both as player 1 and player 2. Because the PIMC player does not take dominated actions, when playing against a Nash equilibrium this player achieves the equilibrium payoff, while a random player loses significantly against even an equilibrium player. If an opponent is able to build a best-response against a PIMC player, then the PIMC player is vulnerable to significant exploitation as player 2, while the random player loses -0.5 as the second player, where 0.056 could have been won. Thus, these results present the experimental evidence showing that PIMC is not a good approach in practice for a game like poker. Although it plays better and is less exploitable than a random player, PIMC may lose significantly to a opponent that can model its play.

5.5 Conclusion

In this chapter, we performed experiments on simple, synthetic game trees in order to gain some insight into the mystery of why Perfect Information Monte Carlo search has been so successful in a variety of practical domains in spite of its theoretical deficiencies. We defined three properties of these synthetic trees that seem to be good predictors of PIMC search's performance, and demonstrate how these properties can be measured in real games.

There are still several open issues related to this problem. One major issue that we have not addressed is the potential exploitability of PIMC search. While we compared PIMC's performance against an optimal Nash equilibrium player in the synthetic tree domain, the performance of PIMC search could be substantially worse against a player that attempts to exploit its mistakes. It would also be informative to examine more real games that more completely span the parameter space established by our synthetic trees.

Finally, we have seen that in games like skat that there isn't a single measurement point for a

game, but a cloud of parameters depending on the strength of each hand. If we can quickly analyze a particular hand when we first see it, we may be able to use this analysis to determine what the best techniques for playing are on a hand-by-hand basis and improve performance further.

On the contrary, Watson, you can see everything. You fail, however, to reason from what you see. You are too timid in drawing your inferences.

-Sherlock Holmes

6 Inference

In an imperfect information game, *inference* is the problem of accurately assigning a probability distribution over unobserved variables in the game state. Informally, in a card game like skat, this means we want to be able to guess the cards that an opponent is holding based on the opponent's actions. It has generally been taken for granted, at least in trick-taking card games such as skat and bridge, that inference of this sort is desirable, but there has been relatively little discussion as to why. For instance, of the 55 pages that Ginsberg uses in his journal paper describing his seminal PIMC search-based player GIB [15], less than 1 page is devoted to inference and the issues related to it. In this chapter, we will discuss the inference process as it relates to imperfect information games and to particular techniques like PIMC search in greater detail, as well as presenting inference-related theoretical results and insights.

6.1 Isn't Inference Just Opponent Modelling?

Inference, as we have taken it, is the problem of mapping a series of observed actions taken by a player into a probability distribution over that player's private information. A reasonable and intuitive question to ask is whether inference is nothing more than an attempt to build a model of a particular opponent. Certainly, since inference attempts to derive meaning from an opponent's actions, it is inherently dependent on the opponent's policy for choosing those actions — in other words, her strategy. And if the opponent does not act as we expect, could our inferences cause us to make more mistakes by misleading us?

To answer these questions, we will revisit the simple abstract game tree in Figure 6.1, which we had previously displayed in Chapter 5. In our earlier discussion, we stated that in this game, the minimizing player (who moves second here) can always make the correct play by moving to the right. This is because he has inferred that if world 1 was the true world, surely the maximizing player would have moved to the right and won immediately. But what if the maximizing player is trying to "trick" the minimizer? Perhaps she takes great joy in tempting her foe with the chance of victory, knowing full well he will throw it away under the assumptions of his inference. So let us suppose that she always moves to the left. In the case where worlds 1 and 2 are equally likely, the minimizer will now win half of the time by always moving to the right, which is the best he can do in this case. However, if world 1 is more likely than world 2 — say it is 75% likely to be the true

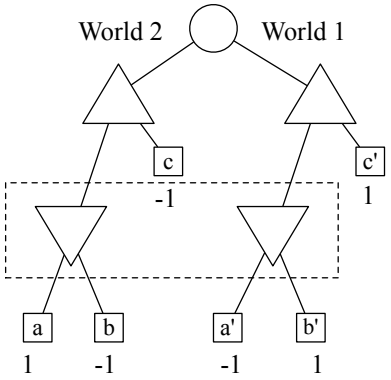


Figure 6.1: A game tree where sound inferences can be made.

world — the minimizer would do better by always moving to the left instead and winning 75% of the time. By insisting on moving to the right due to his inference, he wins only 25% of the time, which is a considerable drop in expected value compared to 75%. However, recall that if world 1 was indeed the true world 75% of the time, then the maximizing player *could* have just won outright 75% of the time by foregoing this little charade and always moving to the right whenver she knows world 1 to be the true world. Therefore, the most the minimizer can actually hope for against an optimal opponent is a 25% win rate, which he achieves by moving to the right (against the optimal opponent or any opponent for that matter).

This discussion leads us to a distinction between what we will call *sound* and *unsound* inference. A sound inference is one that cannot cause a reduction in our expected value assuming a strong opponent. More precisely, in a two-player, zero-sum game, we will say an inference is sound if it is based on the assumption than an opponent is playing a Nash equilibrium strategy. Our discussion concerning Figure 6.1 is an example of sound inference. An unsound inference is based only on the assumption that the opponent is playing an arbitrary fixed strategy. Therefore, so long as we limit ourselves to sound inferences, then the inference process is as generally valid as the widely-used assumption that the opponent is playing an equilibrium strategy.

6.2 Inference and Non-locality

In Chapter 5, we discussed in detail Frank and Basin's concept of *non-locality* [9], one of the two distinct sources of error for PIMC search. This concept is in fact very closely related to the problem of inference. As stated by Frank and Basin, non-locality arises because the opponent may be able to use their private knowledge to direct the play towards sections of the game tree that are more favorable for her. This is akin to saying that moves convey information, using reasoning of the form: "X cannot be true, because if it were, the opponent would have played differently." Inference is the process of capturing this information conveyed by opponent moves, and by the same token, being

aware of the information that our own moves convey to the opponent.

In their discussion of non-locality, Frank and Basin conclude that the problem is not specific to PIMC search in its particulars; rather, non-locality has the potential to arise for any move-selection algorithm that considers only the subtree of the current state (or even current information set). We will call such an approach a *local* solution method. The only way to guarantee correctness is to use solution methods that consider the game tree of an imperfect information game in its entirety; we will term such approaches as *global* solution methods. However, as we have previously established, applying a global method to a large real game is a daunting endeavour. In games like skat, where end-game positions at least appear so much smaller and simpler than the full game, we would like to be able to harness the computational simplicity of local methods. Inference can help us to do just that.

6.2.1 Using Inference to Solve Subgames

Suppose we have a magical black-box that can, at any stage of an imperfect information game, generate an accurate probability distribution over all the hidden (i.e. non-public) information in the game. We will show that we could use such a box to find optimal strategies for subtrees of a game G without the need to consider the entire game-tree of G.

Definition 6.2.1. *Let G be a finite imperfect information extensive form game. Let H be a history of actions taken by the players in G. We designate H as a* public move history *if all of the actions in H were observed by all of the players in G.*

Definition 6.2.2. *Define BOX(G, H, σ) as a function, that, given a game G, a public move history H leading to a set of possible states S, and the strategy profile σ for all players in G, generates $P(s)$ for all $s \in S$ as the correct a posteriori probability distribution over all possible current states S.*

Theorem 6.2.1. *Let G be a finite imperfect information extensive form game and let S be the set of states in G that are consistent with an arbitrary public move history H. Let σ be the strategy profile used by the players in G that generated H. Let G' be a game that begins with a chance node that generates initial game states according to the distribution produced by BOX(G, H, σ) and is otherwise identical to the subtree of G rooted at S. Let σ' be a strategy profile for playing G'. Then the expected payoff for all states in G' under $sigma'$ will be the same as the expected payoff of the corresponding state in G under σ'.*

Proof. Let S be the set of possible states in G. Together, σ and the public move sequence $H = m_1 \dots m_n$ define a conditional probability distribution $P(s|m_1 \dots m_n, \sigma)$ for all $s \in S$ that are consistent with H. Since we have stated σ is fixed, the entire game tree prior to S may be condensed to a single state L where the chance player is to move, and acts according to the distribution $P(s|m_1 \dots m_n, \sigma^s)$ — by definition, this is precisely the distribution produced by BOX(G, H, σ).

Let G' be a game that begins with L and is then followed by all game subtrees from G that are rooted at S such that there is an edge e from L to some $s \in S$ in G' iff the chance outcome represented by e combined with the move sequence $m_1 \dots m_n$ would lead to the same state s in the original game G. Then the expected value for any strategy profile σ' for playing G' will be the same as the expected value for σ' when used in the corresponding subtree of G. Therefore, at each state s' from G' the expected value is same as for the matching state s from G. □

At first glance, we may not have gained much from this theorem, as it depends on our magical black-box. Of course, if we have σ, constructing our box requires no particular special magic but only a straightforward application of Bayes' rule; however, typically σ is not easily available, particularly the components belonging to the opponents. The key insight here is that we only care about σ insomuch as it helps us generate our probability distribution over states, $P(S)$. If we could somehow estimate $P(S)$ directly, then we need not care about obtaining σ at all. The dimensionality of the space of possible states is typically far smaller than for the space of possible strategies; therefore, $P(S)$ may be considerably easier to approximate than σ itself.

6.3 Inference in PIMC Search

We have seen in the preceding sections how accurate inference can enable the use of local methods for move selection without sacrificing guarantees of theoretical correctness. However, that discussion assumed the use of optimal move selection methods. As we have seen in Chapter 5, PIMC search is not an optimal method. A good inference engine solves "half" of the non-locality problem for PIMC search, in that it can take into account information leaked by opponent moves prior to the current state. However, it will remain unaware of information leaked by its own future moves (and those of the opponents), and the problem of strategy fusion remains the same as before. Therefore, it cannot be taken for granted that accurate inference will improve the playing strength of PIMC search. Instead the question must be examined empirically.

6.3.1 Inference in Kermit

In Chapter 4, we presented results concerning the playing strength of our program, Kermit. In those results, we explicitly tested versions of the program with and without its inference module. We reproduce the relevant sections of those results here to discuss them in greater detail.

First, we note that the inference module substantially improves Kermit's performance in self-play. This entry should represent an upper-bound on the performance gain due to inference, since Kermit has available the perfect model of itself. Most of the gain comes from the defenders modelling the soloist, and it is not hard to argue as to why. In particular, a great deal of information is conveyed by the soloist's choice of game-type, especially as to which suit is trump. Without inference, PIMC search will examine worlds at uniform random — which means from a defender's point

Table 6.1: Re-visiting 2-way tournament results that highlight Kermit's inference performance. S and D indicate that Kermit infers cards as soloist or defender, resp. The NI program version does not do any inference.

Type	Name	Point Avg.	Std.Dev.	#
Cardplay	Kermit(SD)	**996**	50	1600
	Kermit(NI)	779	54	1600
Cardplay	Kermit(SD)	**986**	51	1600
	Kermit(S)	801	53	1600
Cardplay	Kermit(SD)	**861**	53	1600
	Kermit(D)	820	54	1600
Full Game	Kermit(SD)	**1188**	30	4800
	XSkat	629	26	4800
Full Game	Kermit(NI)	**1225**	30	4800
	XSkat	674	27	4800

of view, his partner could easily hold more trump cards than the soloist. Furthermore, in any world where this is the case, it is typically easy for the defenders to win the game with nearly any move, which makes it difficult for them to differentiate move quality. In reality, of course, the soloist chose a particular trump suit because she is holding more than her share of those cards, and this is reflected in Kermit's inference.

Good performance in self-play is not enough to validate Kermit's inference, so we also tested Kermit's non-inference version against the rules-based system XSkat. Here we see that the inference has little effect on Kermit's performance; the score differential between Kermit and XSkat remains nearly the same. Both players have lower scores overall because the inference-equipped Kermit plays better on defense, which tends to lower the average score. In some sense this is a positive finding, as it suggests that Kermit's inference is *sound*. XSkat is a relatively weak player which Kermit already dominates with or without inference; therefore, to improve performance in this setting, most likely unsound inference would be required. Combined with Kermit's strength playing online on ISS, this indicates that Kermit's inference is beneficial against stronger players and at least does not mislead the program against weaker players.

6.3.2 Considerations in Inference Design

Even if we have a good method of attaching probabilities to individual worlds, there are some practical subtleties to using inference for PIMC search (or any other local solution method that uses inference to sample worlds) that bear mentioning. Ideally, PIMC search would exhaustively examine all worlds, and weigh the result of each world's analysis by the world's probability. When the state space is still large, this approach is not feasible if the analysis of each world is time-consuming, as is the case when using alpha-beta search on perfect information worlds. The next solution, then, is to build the complete a posteriori probability distribution over all worlds and sample as many worlds as

possible from that distribution. However, if the cost of assigning a probability to individual worlds is too high, this process may still be infeasible, especially near the start of a game when millions of different worlds are possible; this was certainly the case for our player Kermit. One approach then is to sample worlds uniformly and then weight the results of examining those worlds by their probability, throwing out worlds with zero probability and re-sampling if necessary. A downside to this approach is that if there are only a few worlds with high probability and many with a low but non-zero probability, we are unlikely to ever examine the most interesting worlds. A good compromise, and the approach used in Kermit, is to sample a sizeable subset of the possible worlds at uniform random, and then build a probability distribution over this subset, which can then be sampled for worlds that will actually be searched.

6.3.3 Learning Features for Inference

In Kermit, we took the approach of learning features offline for use in inference. In general, we are attempting to store probabilities of the form $P(f|C)$, where f is either a complete perfect information world or perhaps some feature of interest describing a world (for example, the number of cards a player holds in a specific suit) and C is an information context. It is immediately apparent that C should generally consist of the public move sequence, or at least some subset of it that we deem to be most relevant. A key question with which we grappled was whether aspects of a player's private information should be included in C. For instance, say we wish to infer whether or not a player holds the Ace in a particular suit (say ♣). Should we store conditional probabilities of the form $P(has(\clubsuit A)|M, \clubsuit(h))$, where M is the move history and $\clubsuit(h)$ is the constellation of cards that the player performing the inference holds in the ♣ suit? Intuitively at least, it seems natural that our own card constellation might tell us something about the way the opponent will play her cards.

It turns out that in fact we never need consider private information in learning these conditional probabilities. It suffices to store probabilities conditioned on public information; at run-time, private information can be used to simply knock out impossible worlds and the remaining non-zero probabilities need only be renormalized. We show this is true by proving the following theorem.

Theorem 6.3.1. *Let G be a game. Let p_i be a player in G who makes move m in the information set I_i. Then $P(I_i|m, \sigma_i)$ is the probability that p_i is in information set I_i given that we observe his move, m, and his fixed strategy, σ_i. Now let x be private information which was not known to p_i in I_i, but for which $P(x|I_i)$ is known. Then $P(I_i|m, \sigma_i, x) = P(m|I_i, \sigma_i) \cdot P(\sigma_i) \cdot P(x|I_i) \cdot p(I_i)/C$, where C is simply a normalizing constant.*

Proof. A straightforward application of Bayes' rule yields $P(I_i|m, \sigma_i, x) = P(m|I_i, \sigma_i, x) \cdot P(\sigma_i|I_i, x) \cdot P(x|I_i) \cdot P(I_i)$ in the numerator (as always, the denominator is just a normalizing constant). We may now simplify some terms due to conditional independencies. By definition, σ_i maps information sets directly into action probabilities. Therefore, m depends uniquely on σ_i and I_i, and by definition of conditional independence, we may rewrite $P(m|I_i, \sigma_i, x)$ as $P(m|I_i, \sigma_i)$. Since p_i's

strategy is fixed, it does not depend on his information set nor on private information unknown to p_i, so $P(\sigma_i|I_i, x) = P(\sigma_i)$. This is simply a prior belief that p_i plays strategy σ_i. Therefore, with these simplications we now have $P(I_i|m, \sigma_i, x) = P(m|I_i, \sigma_i) \cdot P(\sigma_i) \cdot P(x|I_i) \cdot P(I_i)/C$ as required. □

The key aspect of Theorem 6.3.1 is that the private information x only appears in the numerator once, in the term $P(x|I_i)$. In the special case where $P(x|I_i) = 0$ for "inconsistent" information sets (e.g. we know our own cards, so therefore the opponent cannot be holding any of them) and is otherwise uniform, this has the effect of zeroing out the probability of impossible information sets and renormalizing the rest. For features that depend directly and uniquely on information sets, it is straightforward to see that Theorem 6.3.1 holds as well. This insight is a critical one for the efficient offline learning of world and feature probabilities, since including private information (such as a player's own cards) in an information context exponentially bloats the number of conditional probabilities that need to be estimated

6.4 Conclusion

In this chapter we discussed the issues related to the problem of inference in an imperfect information game. We have presented some theoretical inference properties, as well as practical considerations when using inference in our player Kermit. We suggest that on the whole, considering inference as an explicit problem in the domain of imperfect information games gives rise to a useful mental framework in which we can separate a game into two distinct components: those actions that have happened *prior* to the game's current state, and those that will happen after it. Inference is the glue that allows us to bind this past and future together.

Know your enemy and know yourself, and you can fight a hundred battles without disaster.

-Sun Tzu

7

Real Time Opponent Modelling in Skat

Opponent modelling is the problem of predicting the actions of other agents in an adversarial environment. Classically, the general implicit approach to opponent modelling has been to assume that all players are attempting to play a *Nash equilibrium* — a game-theoretically optimal policy that guards against the worst case. Such an assumption is critical to the alpha-beta minimax search techniques that have been applied with great success to perfect information games like chess and checkers.

However, when adversarial environments become more complex, notably through the inclusion of stochasticity and imperfect information, it becomes more important to consider different opponent models for two reasons. The first reason is that optimal strategies in these environments are often overly defensive — they guarantee that we cannot lose much, but also that we cannot gain much either, no matter the actions of the opponent. In settings such as competitive tournaments, it is often not enough to play well against strong opposition; potential tournament winners must also maximize their gains against weaker, more exploitable competitors. The second reason is that in imperfect information environments, it is often helpful to infer hidden state variables based on actions of other agents. For certain types of inference (notably *unsound* inference as we discussed in Chapter 6, this requires an accurate model of how agents behave in the environment. Thus, an appropriate opponent model allows for an agent to explicitly exploit opponent weaknesses, and to improve its own decision making through superior inference of hidden information.

In this chapter, we describe a simple post-processing analysis technique that can be used to assess an agent's general decision quality in certain classes of environments. We then apply this technique to skat, and show how our agent is able to appropriately adapt its play against varying strengths of opponents. Most importantly, this adaptation is fast enough to be performed in real time and requires only a handful of encounters with a particular opponent in order to be effective.

7.1 Background and Motivation

In Chapter 3, we mentioned how Richards and Amir [32] describe a method of inferring letters on the opponent's rack in the game of Scrabble in order to bias letter distributions used in simulations. Equipped with this inference technique, Quackle, which at the time was the strongest Scrabble program and had previously beaten a former human World-champion, achieved a substantial 5.2

point score advantage over the original Quackle version. Similarly, in our own work in Skat, we saw in Chapter 4 the positive effect that inference had on Kermit's performance. However, in both these cases, the inference was a result of the player modelling itself, rather than explicitly tracking the behaviour of an opponent.

We also discussed in Chapter 3 recent advances in opponent modelling in the game of poker by Johanson and Bowling [17] and Ganzfried and Sandholm [13]. The work by Johanson and Bowling has the advantage of being robust, in that it guarantees a good trade-off between an agent's ability to exploit the opponent and the agent's chance of being exploited itself, but requries a large number of interractions and is too slow for real time use. The work by Ganzfried and Sandholm has fewer guarantees but is faster. However it still requires approximately one-thousand interractions or more to show any positive result. At a live skat tournament, players will typically stay at the same table for one list of 36 games before moving on to a new table with new opponents. Therefore, requiring thousands or even hundreds of games for modelling purposes is much too slow for winning tournaments.

Furthermore, in skat we have strong evidence that opponent modelling is particularly important to achieving a high overall game score. We have mentioned how on our online skat server, Kermit maintains a very respectable list average of 931. However, david, the top human player on ISS has a list average of 1324. This is because david plays many games against weak players like XSkat. What's more, as a result of playing against such opponents, david's behaviour is quite different from Kermit's. On average, david plays as the soloist in a staggering 59% of his games, whereas Kermit plays roughly its "share" of soloist games at 32%. It seems evident that david is able to adjust his bidding to account for the general weakness of his opponents. In this chapter, we will discuss methods to enhance Kermit to exhibit similar behaviour.

7.2 Methodology

In his seminal work on building an expert-calibre computer bridge player, Ginsberg remarks that both human bridge experts and his program GIB have an extraordinarily low "double-dummy mistake rate" [15]. What he means by this is that if one were to compute the *perfect information value* of the game — the final resulting game value under the assumption that the location of all cards is known to all players, and all players play perfectly from that point onwards — that value would rarely change following any play by a human expert or by his program.

Ginsberg does not dwell on this observation, other than to consider it as supporting evidence that his program GIB has reached or surpassed human expert level playing strength. For us, this observation will form the foundation of our modelling approach, which we describe below.

Algorithm 2 PIPMA post-game update procedure for a single player. Winning positions have value 1 and losing positions, value -1; we assume here that there are no other values.

Require: completed episode E, agent p
  for each state s in E for which p is to move **do**
    $Win \leftarrow \emptyset$
    $Lose \leftarrow \emptyset$
    for each action a available in s **do**
      compute value(a)
      {Returns the Perfect Information game value of selecting a at s}
      if $value(a) = 1$ **then**
        $Win \leftarrow Win \cup \{a\}$
      else
        $Lose \leftarrow Lose \cup \{a\}$
      end if
    end for
    if $Win \neq \emptyset$ AND $Lose \neq \emptyset$ **then**
      $mistakeOpps(p) \leftarrow mistakeOpps(p) + 1$
      {p's opportunies to make mistakes, maintained globally}
      if p's chosen action in $s \in Lose$ **then**
        $mistakes(p) \leftarrow mistakes(p) + 1$
        {p's total mistakes made, maintained globally}
      end if
    end if
  end for

7.2.1 Perfect Information Post-Mortem Analysis

We will refer to our approach as Perfect Information Post-Mortem Analysis, or PIPMA. The procedure is intended to be called at the conclusion of each episode in an environment of interest and to incrementally update a *mistake rate* for each agent involved in the episode. This is done by examining each state in which an agent acted and computing the perfect information value of that state before and after the agent's move. If there is a difference, the agent's move is flagged as a mistake. It is also necessary to determine whether it was even possible to make a mistake, which in the worst case, requires examining every available move. States where a mistake was not possible are discarded from the analysis. The agent's final mistake rate is defined as the ratio of mistakes to mistake opportunities. Pseudo-code for the PIPMA process is shown in Algorithm 2.

We note that PIPMA requires two assumptions about the environments to which it can be applied. The first is that by the end of each environmental episode, all previously hidden information from the environment must have become public for all agents. The second is that perfect information values must be relatively easy to compute for all states in the environment. These assumptions hold in most trick-taking card games, such as bridge as studied by Ginsberg, and in skat.

7.2.2 Theoretical and Empirical Properties of PIPMA

PIPMA assumes that the actions it identifies as mistakes in the perfect information version of an environment are actually undesirable in the original imperfect information setting and should be

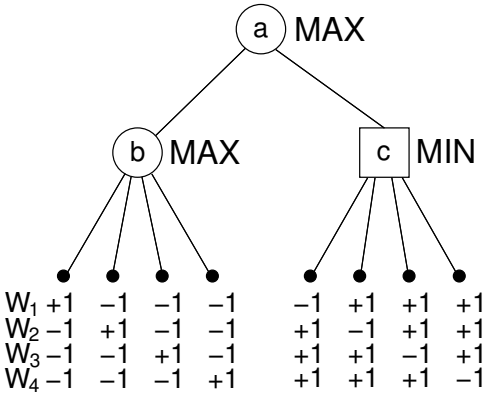


Figure 7.1: A synthetic zero-sum game in which PIPMA flags the optimal move (a-c) as a mistake. The payoffs at the leaves in each world W_i are given in view of player MAX.

avoided by rational players. It is simple to show that there is no theoretical guarantee that this will be the case. In fact, PIPMA can, in principle, consider an optimal move to be a perfect information mistake. We demonstrate this through means of the example shown in Figure 7.1.

In this example, there are two players, MAX (represented by circles) and MIN (represented by squares), who are playing an imperfect information game. There are four possible worlds, determined secretly at the start of the game. MAX is to move at the root and can go either right or left. If she goes left, then she is to move again. Now she has four options, each one of which wins in exactly one distinct world and loses in the other three. Because MAX does not know which is the true world, she must simply pick a move at random, and achieve a payoff of 1 with probability 0.25 and -1 otherwise. Alternatively, if MAX goes right at the root, she can foist the guess onto MIN, and therefore achieve the positive payoff with probability 0.75. Clearly, then, MAX's best move is to go to the right. However, from a perfect information perspective, where all players know the true world, going to the right is a mistake for MAX, since MIN has no need to "guess" in this game's perfect information variant. Therefore, MAX moving right will be flagged as a mistake by PIPMA, even though it is the clear optimal move in the real game.

In spite of this theoretical caveat, it seems that pathological cases like those shown in Figure 7.1 rarely occur, at least in the environments where PIPMA has been examined. Ginsberg shows that in bridge, human world experts have a PIPMA mistake rate of approximately 2% while leading the first trick of the game, and this rate goes steadily down as the game progresses. In skat, we measured card-play mistake rates for a collection of sixty-five randomly selected human players, each of whom had played at least 2000 games on an online skat server. We plot these mistake rates against the players' average *score-per-list* — the standard scoring metric in skat — in Figure 7.2. Although card-play strength is only one aspect of playing good skat, the general trend is clear: higher-scoring players tend to have lower PIPMA mistake rates. We also show skat data in Figure 7.3 indicating how PIPMA mistake error declines as a function of the number of tricks played so

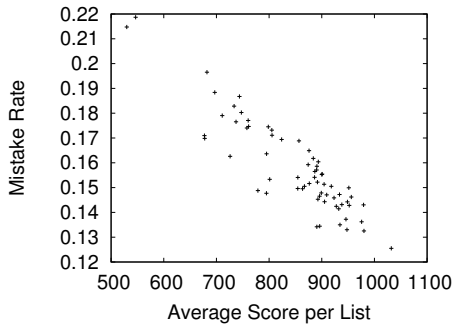


Figure 7.2: PIPMA mistake rate plotted against average score-per-list for sixty-five human skat players.

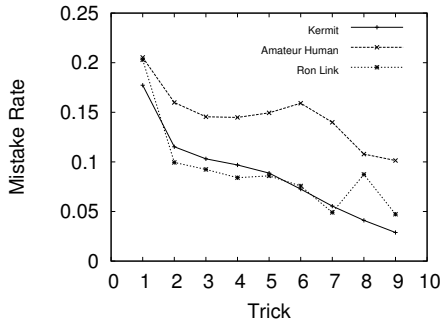


Figure 7.3: PIPMA mistake rate plotted as a function of the current trick in skat for three players. Kermit is a computer player, Ron Link is a high-ranked human expert and the third player is an arbitrary human of roughly average strength.

far. On this plot, we show data for three players: Kermit, Ron Link, who is a high-ranked human expert, and an arbitrarily selected amateur human player. Although overall mistake rates appear to be higher in skat than in bridge, we see a trend similar to Ginsberg's observation, namely that the mistake rate declines as the game progresses.

Finally, we note that PIPMA is relatively difficult for a canny human (or other opponent-modelling computer) player to exploit, so long as cases such as described in Figure 7.1 are indeed rare (and it is likely inadvisable to use PIPMA at all in environments where they are not). In order to be unduly "underestimated" by PIPMA, a player must intentionally throw away games that, from a perfect information perspective, she has already won. It seems exceedingly unlikely that forfeiting guaranteed wins frequently enough to cause PIPMA to make a serious misestimation of a player's skill could result in a significant long-term advantage. However, this is true if PIPMA flags only mistakes that change the game value from a win to loss. If, for example, PIPMA considers a move that loses points according to some heuristic function — say, failing to maximize tricks taken in bridge beyond what is needed to win the contract, or failing to maximize card points in skat — to be a "mistake," then it could become vulnerable to an exploitative opponent. On the other hand, if most of these mistakes are genuine, flagging them with PIPMA could decrease the amount of data

needed to form an accurate opinion of the opponent's mistake rate. In the remainder of this work, we restrict ourselves to considering pure win/loss mistakes, but considering heuristic mistakes as well could lead to interesting future work.

7.2.3 Applying Perfect Information Post-Mortem Analysis to Skat

In this section we describe how we use the PIPMA technique described above to add a real-time opponent modelling component to our computer skat player Kermit.

Recall in skat the asymmetry between the soloist and the defenders. The soloist has the potential to gain many points but can also lose points as well, so it is an inherently risky position. The defenders can win a small number of points if they defeat the soloist during card-play, but cannot lose points in any way. Therefore, playing defense is very safe, but does not allow for the potential of large rewards. Recall also that Kermit uses a simple thresholding approach to decide if it will continue to bid in hopes of winning the auction and becoming the soloist, or if it will pass and resign itself to a defense role. Kermit continues to bid if its perceived winning probability as soloist is greater than the fixed threshold of 0.6.

Our new program, which we will refer to as the Count (due to the program's manic obsession with counting points and moves of the other players, much like the Sesame Street muppet of the same name), is identical to Kermit, except that after every hand, it post-processes the card-play moves of all players involved using PIPMA and updates their mistake rates. This post-processing takes approximately one second after each game, and is therefore practical even for real-time play against humans. The Count then uses this information to dynamically adjust its bidding threshold during the bidding phase of future games.

The basic intuition behind this threshold adjustment is simple: bid more aggressively against weak players (players with a high mistake rate), and less aggressively against stronger ones. The problem that remains is exactly how much to adjust this bidding threshold based on the observed mistake rates of the opponents. To gain insight into this decision, we constructed a synthetic skat player that is directly defined in terms of its mistake rate m as follows. Whenever the player is to move, it is explicitly informed of the true world, and computes the sets of winning and losing moves for the current (perfect information) position. Then, assuming both sets are non-empty, with probability $1 - D(m)$ the player chooses a winning move to play, using Kermit's PIMC search procedure (and thus for this purpose ignoring that the player knows the true world) to break ties if necessary. $D(m)$ is a function that maps the player's global mistake rate to a local one that is dependent on the current stage of the game — this is to account for the fact that in skat, players make many more mistakes early in the game than they do later. With probability $D(m)$, the player is *forbidden* from selecting a winning move and must play a losing move (e.g. make a mistake) instead, again using PIMC search to break ties among candidate losing moves. We chose $D(m) = m + 0.07$ for the game's first trick and $D(m) = \max(0, (m - (0.01(trick - 1))))$ for subsequent tricks. These

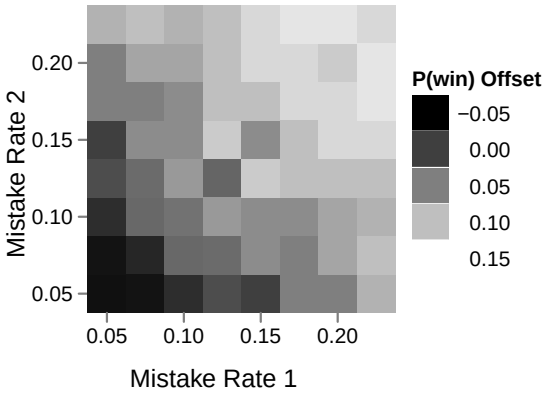


Figure 7.4: Synthetic tournament results displaying the average P(win) estimation error in Kermit's bidding-phase hand evaluator, plotted as a function of the mistake rates of Kermit's two opponents.

choices were based on the approximate observed rate at which Kermit's own mistake rate declined as a function of the current trick, and so that empirically the synthetic player's global mistake rate ends up being very close to m.

We then played a series of games between Kermit and synthetic players of varying strength. For each series, we record the average error in Kermit's bidding evaluation function over all games in which Kermit played as soloist, where the error for an individual game G is defined as follows:

$$\text{error}(G) = \text{win}(G) - \text{pWin}(G) \tag{7.1}$$

win(G) is 1 if Kermit won in G as soloist and 0 otherwise, while pWin(G) is the winning probability for G as computed by Kermit's evaluation function. The results of these experiments are shown in Figure 7.4.

$$[t]\text{offset}(p_1, p_2) = \begin{cases} 0.1 - m(p_1, p_2) & \text{if } m(p_1, p_2) \leq 0.1 \\ 0.05 - 0.5m(p_1, p_2) & \text{otherwise} \end{cases} \tag{7.2}$$

Note that, while useful, these synthetic results do not take account of more complex factors, such as a player's potential to win extra points as a defender due an opponent's high error rate. However, using these as a starting point, we performed some empirical tuning and ultimately chose the formula shown in Equation (7.2) to use for the Count, where $m(p_1, p_2)$ is simply the average of the individual mistake rates of the two opponents, p_1 and p_2. Recall that this is an offset to the Count's bidding threshold, so that a positive offset increases the threshold, making the program more conservative, while a negative offset decreases the threshold, making the program bid on riskier games.

Table 7.1: Tournament results between players, all performed on the same set of 2000 deals. In each match, the column player played against two clones of the row player. We report the average score-per-list difference (row-column) and score-per-list (standard deviation in brackets) for row and column players in each match-up.

row \ col	Kermit			Tracker			Tracker-rm36		
	r-c	row	col	r-c	row	col	r-c	row	col
Mistake0.03	379	874(46)	495(55)	344	812(47)	468(55)	262	756(48)	494(50)
Mistake0.10	-213	634(48)	847(48)	-90	693(48)	783(52)	47	684(48)	637(53)
Mistake0.25	-638	460(52)	1098(45)	-726	460(50)	1186(48)	-498	468(49)	966(56)
XSkat	-741	648(42)	1389(47)	-826	612(36)	1438(39)	-662	594(31)	1256(63)

row \ col	theCount			theCount-rm36		
	r-c	row	col	r-c	row	col
Mistake0.03	108	756(48)	648(48)	133	761(48)	628(46)
Mistake0.10	-161	660(48)	821(48)	-144	662(49)	806(48)
Mistake0.25	-618	511(50)	1129(52)	-564	493(52)	1057(53)
XSkat	-818	630(39)	1448(52)	-808	623(39)	1431(39)

7.3 Experimental Results

7.3.1 Computer Player Tournaments

To assess the strength of our PIPMA modelling approach in a real game environment, we ran a series of skat tournaments between computer players. Our goal is to compare the performance of three different *candidate* players: Kermit, the Count and a player we call Tracker. Kermit is the static, non-adaptive skat player described in Chapter 4. The Count is the modification of Kermit that uses PIPMA to adjust its bidding behaviour, as described in Section 7.2. Tracker is a "naive" adaptive player also based on Kermit that tracks only its own performance as soloist in the context of its current opponent match-up: after each soloist game played, Tracker updates its perceived error in its bidding evaluation function according to Equation (7.1) from Section 7.2. It then uses the average accumulated error as an offset to its bidding threshold in future games. Tracker's final adjusted bidding threshold is capped at 0.75, so as to prevent the program from believing that all soloist games are unwinnable (and therefore never playing as soloist again to gather more data) after a particularly unlucky streak.

We note that the card-play strength of our three candidate players is identical. Furthermore, the adaptive bidding of Tracker and the Count is such that against an opponent of equal card-play strength, the bidding of these two programs is nearly identical to Kermit's. Therefore, comparing these programs by playing them against each other would be futile.

Instead, we play each of these candidates against a set of *benchmark players*. Three of these are synthetic players, as described in Section 7.2, using mistake rates of 0.03, 0.1 and 0.25. The fourth benchmark player is the rules-based program XSkat. We have previously shown in Chapter 4 that XSkat is substantially weaker than Kermit. It serves as a useful example of a player that is very

different from the synthetic players and our candidate players.

In our experiments, each candidate played the same set of 2000 skat deals against two clones of each of the benchmark players (although the candidate players were not aware that their two opponents were identical for the purpose of modelling). All experiments were run on computers with eight 2.5 GHz Intel Xeon E5420 CPUs with 16 GB of RAM.

We show results of these tournaments in Table 7.1. The displayed score represents average points earned per list (36 games). The candidate players Tracker and the Count have two types of entries in this table: one with the suffix "rm 36" and one without. The "rm 36" version of each program deletes all of its opponent modelling data every 36 games; the non-suffixed version keeps its data for the duration of the 2000 game run.

Discussion

In opponent modelling, the general goal is to improve performance against some opponents, ideally without losing in self-play or becoming too vulnerable to exploitation by an adaptive adversary. Our experiments against the candidate players were designed to test the first two of these three criteria and we will discuss the results in that light.

Against the Mistake0.1 player, we see little difference between Kermit and the Count; in fact, Kermit performs slightly better. This is not unexpected, because Kermit's own global error rate is very close to 0.1, and therefore this result approximates a self-play match. Since Kermit is tuned to perform well in self-play, the best the Count could hope for in this match-up is to tie Kermit's performance, which it effectively does.

Meanwhile, the Count achieves a modest but, in skat terms, still substantial gain against XSkat, and a dramatic gain against the exceptionally strong Mistake0.03 player. The Count's somewhat weak performance against the Mistake0.25 player was more of a surprise. Although the Count still outperforms Kermit in the long run, this advantage is slight enough that it disappears when the Count is forced to erase its modelling data. This appears to be because although the synthetic player's cardplay is quite weak, it still bids as aggressively as Kermit, meaning that Kermit earns many points on defense and would achieve little by increasing its own bidding aggressiveness (as the Count tries to do). XSkat, while of similar card-play strength to the Mistake0.25 player, is more conservative in its bidding. A follow-up experiment where the Mistake0.25 player's bidding threshold was increased from 0.6 to 0.7 seemed to confirm the role of the bidding behaviour; in this setting, the Count (with "rm36" enabled) exceeds Kermit's list-average by 57 points.

Finally, we compare the Count to the Tracker player. Our objective in including Tracker among our candidate players was to compare the amount of data needed to perform successful opponent modelling by a naive approach against the amount needed by the Count's PIPMA. In nearly all cases, we see that Tracker performs quite similarly to the Count in the long run. However, when it is forced to delete its modelling data every 36 games, the performance of Tracker degrades quickly, becoming

weaker than both Kermit and the Count against all the benchmark players except Mistake0.03, where it is roughly even with Kermit. The Count, on the other hand, becomes only very slightly weaker when it deletes its data. Effectively, Tracker is not able to accurately model its opponents in a short period of time using only end-of-game results whereas the Count gains considerably more information per game by examining the individual moves made by all players. This appears to allow the Count to model its opponents sufficiently quickly inside a 36 game window to make a positive difference in its play.

7.3.2 Online Play

In addition to the tournaments between computer players described above, the Count has been playing on ISS. When we first introduced the Count to online play, it achieved a list average of 1000 points (std. deviation of 18) in its first 14415 games, which was 60 points more than Kermit's average at the time. Since then, the difference between Kermit and the Count has decreased slightly, with the Count dropping to a 963 list average over 134,000 games, though it is still ahead of Kermit's 937 average. The Count's behaviour is quite different from Kermit's, as it plays 38% of its games as soloist, up from Kermit's 32%. This suggests that the Count's modelling approach works well against human players in addition to synthetic ones, although we must be cautious about drawing conclusions in this setting, since so much depends on the make-up of matches. For instance, the more the Count plays with a clone of itself or with Kermit (as often happens if a human invites two computer players to a table), the more its score will converge to Kermit's, which is a likely reason for the Count's slow decline in score. Perhaps the most encouraging result of the Count's modest outperformance of Kermit on ISS is that since humans are typically good at exploitative play, it appears that the Count's PIPMA approach is indeed robust against exploitation in practice, as suggested by our arguments in Section 7.2.

7.4 Conclusion

In this chapter, we proposed Perfect Information Post-Mortem analysis as a simple tool for quickly assessing opponent decision-making abilities in certain classes of environments, notably trick-taking card games. We then used this technique to create the Count, a computer skat player that adjusts its bidding behaviour in real time in response to the card-playing strength of its opponents. We presented experimental results showing that the Count generally performs approximately as well or better — and in some cases, substantially better — as a non-adaptive player against several opponent types. Finally, the Count is able to successfully model its opponents in fewer than 36 games, the length of a standard skat tournament match, making it practical for real-time play against humans.

In pursuing the work we have described here, we have effectively been dealing with two separate questions: first, can we capture some model of the opponent's behaviour, and second, if so, what should we do about it. PIPMA is a relatively general technique that addresses the first of those

questions in environments that are easier to deal with "in hindsight" than they were at the time that actual decisions were being made. This appears to be a fast and robust way of capturing a general concept of "playing strength" in such domains.

The second question, how to appropriately react to a known opponent mistake rate, seems considerably more domain specific. In skat, we were fortunate in that a general approach to this question — adjusting bidding rate — seemed immediately apparent. However, although we performed various experiments to explore how the bidding rate should be adjusted, our final approach is still somewhat ad hoc and could likely be improved in future. In other domains, it is less apparent what the first steps should be in reacting to a known general opponent weakness.

Finally, we note that our focus here has been on opponent modelling in an adversarial setting. However, there are numerous non-adversarial contexts in which a PIPMA-style assessment of agent decision quality could be useful. Take, for instance, a card-game tutorial program of the type suggested by Kemp et al. [20]. Such a system might wish to match a novice human player against computer players of an appropriate skill level. PIPMA provides a straightforward means of evaluating this skill level both for the human and for computer players. In fact, PIPMA may be even more easily applicable in such domains, since an administrating program can have access to full post-game hidden information even if the players involved normally would not. Thus, non-adversarial settings are another promising avenue along which this work could be further explored.

The average Ph. D. thesis is nothing but a transference of bones from one graveyard to another.

-James Frank Dobie

8 The Investigator's Graveyard

During the course of our work to build a strong computer skat player and to understand the issues surrounding that goal, we investigated a number of techniques that failed to live up to our initial expectations for them. For some of these ideas, perhaps this is because they are simply entirely without merit. For others, skat may simply not be the right domain, or perhaps they are lacking some crucial refinement. In this chapter, we will chronicle what we believe to be the most interesting of these techniques.

8.1 UCT Search in Skat

Initially, we had aspirations of extending the work of Schäfer [38], who applied the UCT search algorithm to skat. Compared to PIMC search, UCT has the potential to better address issues of imperfect information. Recall that UCT works by building up a tree of game state nodes, where statistics on move performance are kept at each node. If this tree does not encompass the entire game (as it usually will not), a roll-out module is used to simulate the remainder of the game. For an imperfect information game, UCT's statistics could be kept at the level of information sets, rather than specific game nodes, enforcing the constraint that players must act the same within an information set. This regime still does not capture all aspects of imperfect information; for instance, if we only sample worlds where the player to move has his true hand (as was done by Schäfer and is typically done in PIMC search), then the other players in the UCT tree will implicitly "learn" the hand of the player to move. Nevertheless, this approach is still more imperfect information aware than PIMC search. Additionally, since perfect information game values underestimate the soloist's winning chances, we considered using UCT for the bidding phase as well as cardplay.

In spite of our early optimism, preliminary results by us in combination with Kermit's eventual domination of Schäfer's UCT player on our online skat server caused us to abandon a UCT-based approach. We believe we can provide at least some insight as to why UCT seemed ineffective in the domain of skat.

First, let us consider UCT for bidding. Suppose our player, p_1, is trying to decide whether to *accept* a bid or to *pass*. To do this using UCT, p_1 would sample both moves and begin to build up a tree for the next few levels of information sets — for instance, selecting a game type and then a discard once the bid has been won. The problem, using a sampling approach, is that in skat, the

vast majority of game types and discards are catastrophic, no matter how strong p_1's hand. Using relatively uninformed playouts, it takes UCT an enormous number of simulations to discover that for one very specific game type with a small number of reasonable discards, it actually has a good game as soloist. What's more, because the initial samples on the *accept* branch are so bad, UCT will spend much more time examining the *pass* branch. Initially, this branch will look very good to UCT, for the same reason that the *accept* branch looked so bad; the opponent will lose most game types and thus UCT anticipates earning some nice, safe, tasty defender points. In the event that it is likely the opponent actually has a winnable soloist hand, it will take UCT many simulations to determine that the defender points may not be so likely after all, and then it must return to primarily sampling the *accept* branch. The end result of this is that a player using UCT for bidding does not win the bidding auction very often. This suggests that perhaps UCT is ill-suited for dealing with games that require a very narrow line of play to accurately evaluate certain branches of the game tree.

For cardplay, UCT seems to suffer from a somewhat different issue. Recall that UCT uses a static roll-out policy to select moves deeper in the game tree. While this roll-out policy can vary in strength from being completely random to being highly informed with domain knowledge, it is still assumed to produce weaker moves than the UCT process itself; otherwise, why not simply use the roll-out policy to play the entire game and forego UCT entirely? Therefore, the UCT search paradigm assumes that we are mostly considering high-quality moves for all players near the root of the search, and that deeper in the tree we have a weaker model of how players will act. However, as our results from Chapter 7 have suggested, this model has it backwards when it comes to skat; near the end of the game, players tend to make very few mistakes, and thus are not accurately modelled by a relatively weak roll-out policy.

8.2 Recursive PIMC Search

Recursive Perfect Information Monte Carlo search is a technique briefly suggested by Ginsberg [15] for addressing some of the errors made by PIMC search. The basic idea is simple: instead of using PIMC search to choose moves only at the root of a search, use it to assign moves further down in the game tree as well. Thus, instead of predicting that players will play the optimal perfect information move in each world, we will instead predict that they will play the move that would be suggested by PIMC search in their position. We show pseudocode for this in Algorithm 3.

The major benefit of this recursive approach is that it captures to some degree the concept that moves convey information. We will illustrate this through an example. Consider the miniature skat situation in Figure 8.1. Let us say the only remaining trump card is the ♣J, which is held by the soloist along with the ♠7. Defender 1 is up next to lead, and holds the ♠98. His partner, defender 2, is in the middle and holds ♡A7. Furthermore, since the ♡A is the only high-point card left, let us say that whichever side wins that card will win the game. Let us also say that defender 2 knows which four cards are held by the other players, but does not know how they are distributed.

Algorithm 3 Top level procedure for Recursive PIMC Search

Require: current information set I, set of worlds W that are legal in state I, maximum available time T, maximum time for recursive search t, a global collection $M(w)$ that maps perfect information worlds to moves, maximum recursive depth d

```
for each action a that is possible at I do
    {Initialize values for all actions}
    v(a) ← 0
end for
t ← elapsedTime
while t < T do
    sample some world w ∈ W
    for each action a that is possible at s do
        {applyAction(w, a) returns the world that results when action a is applied to state s}
        w' ← applyAction(w, a)
        recursivePIMC(w', d, t))
        for i from 1 to d do
            w' ← applyAction(w', M(w')
            v(a) ← v(a) + solve(w')
        end for
    end for
    t ← elapsedTime
end while
return argmax(v(a))
```

Algorithm 4 Recursive sub-procedure for Recursive PIMC Search

Require: perfect information world w, recursive depth d, maximum time t

```
if d = 0 then
    return
end if
{I(w) gives the information set to which world w belongs from the perspective of the player-to-move in w}
M(w) ← PIMCSearch(I(w), t)
w' ← applyAction(w, M(w))
recursivePIMC(w', d − 1)
```

From a perfect information perspective, this is a won position for the defenders. Defender 1 is guaranteed to make one trick in ♠, allowing his partner to put on the ♡A. Note that in in this position, playing either of his cards is a winning move for defender 1, since both defeat the ♠7 which the soloist is forced to play. Therefore, if PIMC search was being used to play this position, it would randomly select one of the two cards to play first.

The problem, from defender 2's perspective, is that he does not know exactly which cards defender 1 is holding. In particular, defender 1 might be holding the ♠8 and the ♣J. This position is still a perfect information win for the defenders, and defender 1 could still play either card first; if he plays the ♠8, he knows he is losing the ♠ trick, but will win the last trick with the jack, and will expect his partner to save his ♡A for that trick. However, to defender 2, this situation is indistinguishable from our earlier scenario, where defender 1 holds the ♠98. In that world, he must immediately put on his ace if he sees the ♠8, because the soloist will win the second trick with the

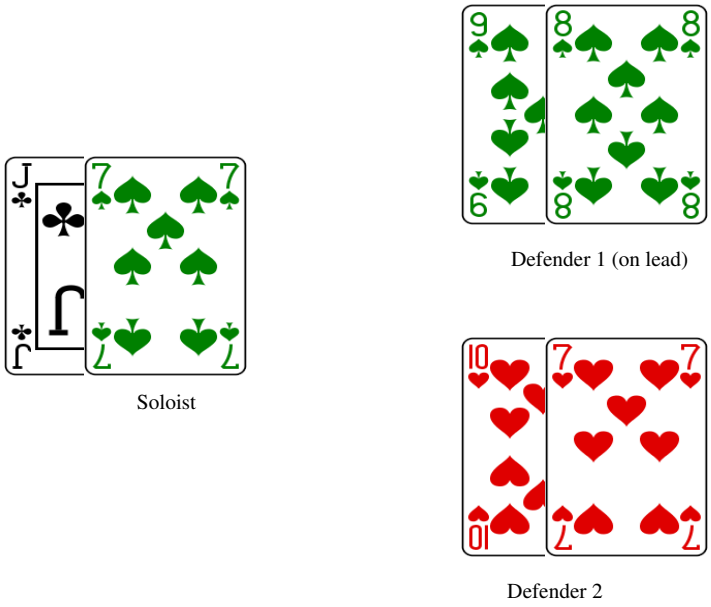


Figure 8.1: A cardplay situation where PIMC search may play an ambiguous move as Defender 1.

jack.

The entire dilemma could be avoided if defender 1 simply played ♠9 or the ♣J — whichever he actually has — first, so that his partner knows what is going on. Leading with the ♠8 is an *ambiguous play* for which defender 2 always has a winning move, but cannot tell which move it is. Our player Kermit, using PIMC search, frequently makes such ambiguous moves, and while they may or may not be the worst mistakes in terms of playing strength, anecdotally they are the most frustrating mistakes for human players. Recursive PIMC search would help solve this problem, as it will show that other players will sometimes make mistakes in the face of ambiguous moves.

The number of required PIMC searches in Algorithm 3 is dWB, where W is the number of worlds examined, B is the average branching factor and d is the depth up to which we will apply the recursive procedure. What's more, B will be larger than for vanilla PIMC search, since we cannot collapse cards that are adjacent in rank and therefore equivalent in the perfect information setting (indeed, the whole point is NOT to do this!). If we model other players as using recursive PIMC search themselves (rather than using standard PIMC search to fix their moves), the number of searches increases to $(WB)^d$. Even in the former, simpler case, this complexity is too high for practical use, especially in the first few tricks when solving worlds is relatively expensive. Nevertheless, to gain some small intuition as to the potential gains of recursive PIMC search, we implemented a depth-1 recursive PIMC search player and tested it against Kermit on a set of games that had been partially completed by human players. The two players used a fixed number of worlds for each search instead of a time-limit. Over 1000 deals where each program played both sides of the deal starting from the 6th trick, the recursive player earned 1.2 tournament points per game (43 points

per list) more than Kermit. This amount is not insubstantial in skat terms. However, it came at a cost of a 100 fold increase in computation time. When the same set of deals was played starting from trick 8, the difference dropped to less than 0.5 points per game. For now, it appears the cost of this technique is prohibitive for the relatively modest gains involved, especially for a system intended to play at human speeds.

8.3 Tracking Bidding Behaviour

In human skat clubs, different players are known for having widely varying bidding behaviour. Some players are quite conservative, content to play defense much of the time and bidding only on very strong games, while others bid much more aggressively. We investigated the problem of modelling this behaviour using real-time post-game processing similar to the PIPMA technique we discussed in Chapter 7.

We begin by observing that by the conclusion of each deal, we know precisely the initial hands h_i of each player i who played defense. We also know the maximum bid that each player made during the auction, b_i. We can therefore use Kermit's "10+2" bidding evaluation to assign a soloist winning probability $P_i(h_i, b_i)$ to the initial hand of player i. Our data then consists of pairs of the form (B, P), where B is a binary variable that indicates whether the player bid or passed and P is the winning probability of the player's hand according to the 10+2 evaluation. If we now assume that the opponent is, like Kermit, a *threshold player* who bids whenever $P > T$ for some threshold T, our data now induces a collection of inequalities on T. These inequalities take the form of $(T < P)$ for each data point (B, P) and $(T > P)$ for each point $(\neg B, P)$. Unless the opponent truly is a strict threshold player (and moreover, one that uses Kermit's 10+2 hand evaluation), some of these inequalities will be contradictory. But we can find a T that minimizes some loss function with respect to these inequalities, for example, a simple hinge loss function where for an inequality $(T < P)$, our loss L is defined by $L = T - P$ for $T > P$ and 0 otherwise. If we discretize T, then this minimization can be done with simple hill-climbing, since the hinge loss function is convex. We graph this loss function as a function of the threshold T in Figure 8.2 for Kermit (using a threshold value of 0.5), XSkat and an aggregate collection of online human players, using 10,000 games to generate the bidding inequalities in each case. We see here, as we might expect, that Kermit's loss function is much steeper than XSkat's and the humans', as well as seeing that XSkat is quite conservative with its bidding in general. We also see that XSkat's overall loss is much higher than that of Kermit or the human players; this is indicative of inconsistent bidding on the part of XSkat, as it frequently misjudges the winning chances of its hands.

In the case of a soloist who picks up the skat, we have a snag, in that there is now no way to extract the soloist's initial hand from the public play history (without cheating, of course). We know the 12 cards that the soloist had prior to her discard, but we do not know which two of the those twelve came from the skat. We could try to estimate probabilities for different possible initial hands,

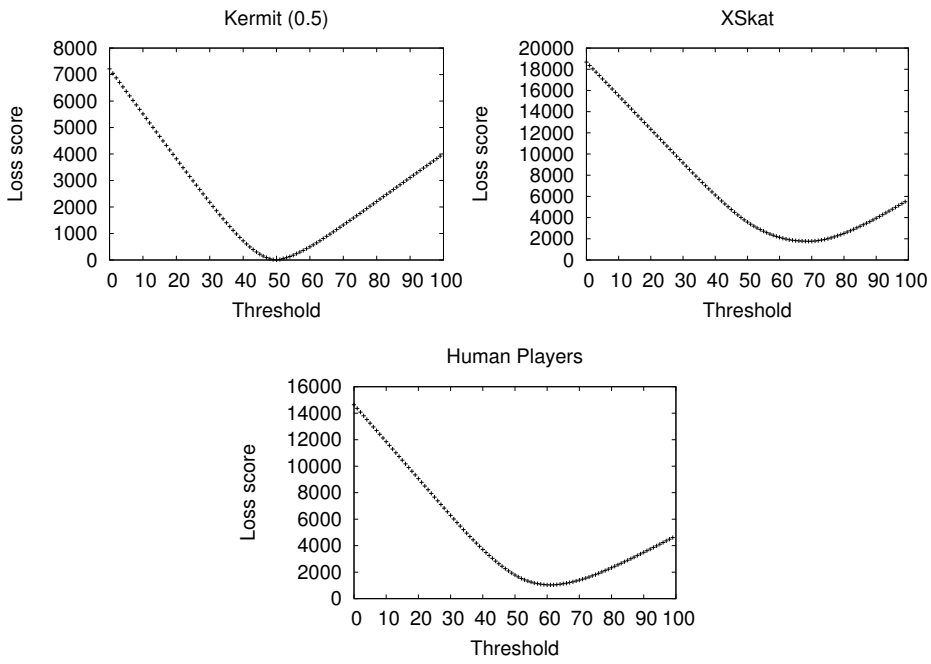


Figure 8.2: Loss value plotted against estimated threshold for Kermit (using threshold 0.5), XSkat and an aggregate collection of online human players.

but this requires a model of how the opponent bids, which is what we're trying to get at in the first place. Moreover, the data that we get in this case is of marginal use, since it results in a single lower-bound inequality. Most of our information is obtained in cases where a player started bidding but then eventually passed, since this will produce two relatively tight inequalities on their assumed threshold T. In practice, we can simply omit recording any data at all for the soloist (except in hand games, where we do in fact know the soloist's initial hand) and still get very fast estimation.

For true threshold players, this estimation is extremely fast, and will nearly always pinpoint the player's true bidding threshold in fewer than 36 games. For other players, such as humans or XSkat, if we assume the player's true optimal threshold to be the one estimated after examining thousands of games, then the average absolute estimation error after using only 36 games is less than 0.02 (recalling that thresholds range from 0 to 1).

Although it appears that this bidding estimation is both quick and accurate, this section sits in the graveyard of this document because we have yet to find a way to use this information to improve playing strength in a significant way. Initially, we had thought to adjust our own bidding behaviour as a response to opponent bidding, but it appears that Kermit's default threshold of 0.6 is quite robust no matter the bidding of the opponents. While bad bidding can be catastrophic for the opponent, it seems to be difficult for us to capitalize on it in the same way that we took advantage of the opponent's cardplay strength in Chapter 7.

8.4 Beyond Threshold Bidding

Kermit's model of threshold bidding — bidding whenever its perceived winning chances as soloist are above some win-probability threshold — appears to be simple, but effective. However, it is a myopic view in that it leaves out factors such as the potential to win points on defense and inferences about what we might find in the skat given the bidding of the opponents. It seems like we should be able to do better.

A slightly more sophisticated model treats bidding as an expected value (EV) calculation that takes opportunity costs into account. More specifically, under an EV model, we continue to bid so long as $pWin_s(B) \cdot winValue - (1 - pWin_s(B)) \cdot lossValue > pWin_d(B) \cdot defenderValue$. $pWin_s$ is the program's perceived chance of winning as soloist and $pWin_d$ the winning chance as defender, both as functions of the current bidding level B. This model assumes that $pWin_s(B)$ is monotonically decreasing in B, while $pWin_d(B)$ is monotonically increasing in B. Strictly speaking this assumption may not be true due to inferences that may be made on the two cards in the skat, but it is certainly true from the perspective of Kermit's 10+2 evaluation function and is a good bet in general.

Kermit's evaluation function already provides us with $pWin_s$, but now we need $pWin_d$ as well. One way to obtain an estimate for this is to sample possible hands for the opponent, and use our trusty 10+2 evaluation to compute the opponent's winning chances, and use $pWin_d = 1 - pWin_{s-opponent}$. In sampling opponent hands, we must take only those that are consistent with the opponent's bidding behaviour so far; this of course requires a model of how the opponent bids. We can assume the opponent bids as Kermit would have, possibly modifying our model based on techniques discussed in the preceding section. In the same way, we can sample opponent hands in combination with the skat to assess probabilities for the cards we may find should we win the bid, weighting the the resulting expected game values appropriately. In the special case where the opponent has not yet bid, we set $defenderValue$ to 0, since we know we will make no points if all players pass.

Unfortunately, testing this EV bidding model in self-play against Kermit resulted in no statistically significant difference in playing strength. We hypothesized that this approach might yield better results against an opponent with more "informative" bidding, such as being very conservative or aggressive, but here too the EV model failed to outperform Kermit's simple threshold bidding. We believe there are two reasons for this. The first is, as we have stated earlier, that Kermit's threshold of 0.6 has proven to be surprisingly robust. The second is that the learned entries of Kermit's 10+2 evaluation function are quite noisy; some are even empty of data and therefore have a default value. In simple models like threshold bidding, the effect of this noise is washed out by averaging over the large chance node from picking up the skat. But for techniques that rely upon more precise values from the evaluation, the effect of the holes in the tables becomes more pronounced. Therefore, we suspect the evaluation itself will need to be re-visited before more complex bidding schemes can

show their promise.

8.5 Randomized Perfect Information Search

In Chapter 7, we saw that in real skat games, good players very often make the correct perfect information move, but not quite always. To model this, we experimented with the idea of replacing perfect information alpha-beta search as the module for evaluating worlds in the PIMC search regime with a randomized perfect information search. The modification is straightforward: at each node, with probability $(1 - \epsilon)$ we play as normal, and with some small probability ϵ, the node becomes a chance node, with the move selected at uniform random. As a result, the evaluation of each world becomes non-deterministic, but PIMC search can sample the same world more than once if need be. The hope is that this approach will help differentiate moves that are certain wins or losses from those that depend on the other players playing a very narrow line of play. Another nice feature of this approach is that it invites straightforward opponent modelling, since ϵ can easily be adjusted for individual players whose mistake rates have been measured.

In spite of these hopes, a preliminary test of this idea showed a slight decline in playing strength both in self-play and against XSkat. We suspect that a uniform model of what mistakes are made and where is too crude. For example, few players would make the mistake of happily putting on their 10 when an opponent has led the Ace in the same suit. Leading the wrong suit, on the other hand, is quite common. Investigating how easily common mistakes can be identified and categorized may lead to better results for this approach.

8.6 Conclusion

In this chapter, we gave an overview of a number of techniques which we investigated in the context of our work on skat but that failed to give noteworthy positive results. Perhaps a big-picture lesson we can take away from this work is that at Kermit's current reasonably high playing strength, there is probably no single enhancement that will result in a large improvement in all settings. It may be more fruitful to target particular aspects and weaknesses of Kermit's play, and only when a number of techniques focused on different relatively rare "problem" scenarios are combined will we see a marked improvement in the full game.

The future will be better tomorrow.
-Former US Vice-President Dan Quayle

9 Lessons Learned and Future Work

In this final chapter, we take a step back to consider some more general insights suggested by the results described in this dissertation, and to propose a few directions in which this work could be extended in future.

9.1 Lessons

There are, we believe, some high-level lessons that manifest themselves across various aspects of the work described in this document. We will discuss these in terms of three themes: the role of local methods and of independence, the importance of understanding success as well as limitations, and the need for testing regimes beyond self-play.

9.1.1 Local Methods and Independence

One of the aspects of imperfect information games that makes them so difficult is that they are fundamentally non-local structures. To guarantee theoretical correctness, the entire game tree must be taken into consideration; this is in contrast with perfect information games, wherein a subtree may be optimally solved without the need to examine any other section of the game tree.

Nevertheless, the success of PIMC search as shown in this document indicates that local methods are still a powerful tool for certain classes of environments. Local methods have significant advantages in computational tractability over global ones and allow us to focus on the "here and now" of specific circumstances. We gained a useful perspective on this property during our brief investigation into using the CFR algorithm to solve skat end-games that we described in Chapter 5. Under the CFR regime, we sampled individual worlds for each CFR iteration. Recall that for CFR, we must sample worlds in which we (the player computing the move) have a hand other than our true hand in order to accurately compute strategies for the opponents. However, if the total number of iterations is relatively small and we do not sufficiently sample worlds where we *do* have our true hand, the playing strength of the strategy computed by CFR suffers drastically. At the end of the day, the information set where we have our true hand is the one that will be used to actually select a move. In skat, it seems that knowing how best to play the hand that we have is much more important than knowing how to play the many hands that we do not (and may not ever) have.

Key to local methods is the power of the independence assumption: the belief that various as-

pects of the game tree can reasonably considered to be independent of one another. Although this assumption is rarely completely justified, in many cases it seems to be "true enough" to yield good results. Not only is it the key assumption behind PIMC search (that worlds themselves are relatively independent), but it is important for inference. If we wish to know whether a player holds the ♡A, say, we might consider the most relevant moves to be the ones where ♡ were played. This is effectively saying that the probability that a player holds the ♡A is independent of moves that did not involve any cards in the ♡ suit. Identifying and exploiting such structural independencies is therefore of considerable interest.

9.1.2 Understanding Suboptimality

PIMC search is an algorithm with no hard theoretical guarantees; in fact, it is easy, at least in skat and bridge, to devise examples where PIMC search will select the worst possible move. Yet in practice, it performs well, a mystery we investigated in Chapter 5. The main lesson here is that even provable suboptimality is not sufficient cause to discard out of hand an empirically effective method. Instead, perhaps a new paradigm is needed to describe the suboptimality of game-playing algorithms. We may need to talk about specific game tree structures and local independencies (again, highlighting the importance of the search for independence) at a more granular level.

Similarly, our work in Chapter 7 suggests that a better understanding is needed of the relation between an imperfect information game G and its perfect information variant G'. In general, not much can be said about the connection betwen G and G', as a strategy for playing one may be arbitrarily bad in the other, yet it seems misleading to say there is no connection at all. In skat, it is clear that there is such a connection between these game variants, and an extraordinarily deep one. It seems incredible that coincidence alone could explain the fact that much of the time, good players will play the move in G that would have been optimal in G'. A better understanding of this connection may help us to generalize game-playing techniques across different classes of environments.

9.1.3 Self-play is Not Enough

When considering a new enhancement to a computer game player, it is tempting to consider the main standard of improvement as an increase in performance against the player's previous incarnation. In fact, in the domain of imperfect information games, it is important to test performance against a variety of players of different playing strengths. In Chapter 6, we saw how sound inference may not show a performance gain against certain players; testing against different players is also important to ensure the inference is not an "over-fit" of the program's model of itself. In Chapter 5, we discussed how in some game classes, the greatest weakness of PIMC search may not be its performance against an equilibrium, but rather its potential exploitability. And in Chapter 7, we introduced a method for constructing synthetic players with an adjustable playing strength to study techniques for opponent modelling. An interesting point about this last approach is not only did we create weak synthetic

players, but it was also possible to create "super optimal" synthetic players that were *stronger* than our program, or any known program. The big message here is that when working with imperfect information games, one should keep in mind that performance evaluation can be deceptively tricky and may require a number of different evaluation techniques.

9.2 Future Work

Throughout this document, we have occasionally made mention of possible improvements and future work, often in the conclusions of the preceding chapters. In this section, we will add some further directions in which the work presented in this dissertation could be extended in the short and mid-term.

9.2.1 Identifying Mistakes

We saw in Chapter 7 how a simple post-game analysis technique can be used to quickly gain a general impression of a player's cardplay skill. The next step in this regard is to examine how quickly we can estimate the types of mistakes that an opponent makes. If, for instance, an opponent's mistake rate is considerably higher in null games than in other game types, we could adjust our bidding accordingly. Furthermore, there is potentially great value in pinpointing where players are likely to make specific cardplay errors. In Chapter 8, we discussed the idea of a randomized alpha-beta search for use in PIMC evaluations. A better understanding of where players make their mistakes could not only enhance this technique in terms of plugging the "gaps" in PIMC search's play, but could also allow us to exploit opposing players in the cardplay phase as well as in bidding. For example, human players on ISS have anecdotally reported that XSkat can be counted on to make specific cardplay mistakes; we would like for a computer player to be able to exploit these mistakes as well. Finally, a better model of how players make mistakes would be useful in improving the synthetic players that we used to evaluate performance in Chapter 7, given how earlier in this chapter we highlighted the value of testing against opponents with scalable playing strength.

9.2.2 Inference from Cardplay Moves

Currently, Kermit's inference stems entirely from bidding and game type selection decisions made by the other players. Although we believe these are the most informative moves in a typical skat game, information is undoubtedly conveyed by cardplay moves as well. We did investigate the use of simple cardplay features for inference at some point in this work, but were unable to find features with a significant effect. This may just be a feature-search problem, but there are other approaches that could be taken as well. One such approach would be to use Kermit's PIMC search module to examine past opponent moves and determine if the opponent would have acted as observed in a given hypothetical world. This is likely to be computationally impractical much like the recursive PIMC search discussed in Chapter 8. However a variation on this theme is to examine opponent

moves in the context of a perfect information search, and use the opponent's known mistake rate as discussed in Chapter 7 to weight worlds where the opponent did or did not play the expected optimal perfect information play. This approach would require a very fast perfect information solver, but we are aware of work by Furtak and Buro that may achieve exactly that [12].

9.2.3 Iterative Data Generation

When constructing Kermit, we were fortunate to have access to a large volume of human game data. This was critical for building Kermit's 10+2 hand evaluator, which has been quite effective in practice. However, human data is not always easily available, and furthermore, even our existing data has a downside: we tend not to see negative examples (for example, the result of playing grand games with weak hands), and certain rare card configurations do not show up in the data (having 9 or 10 trump cards is very good, but we never see it, so our evaluation assumes it is very bad). An interesting question is whether, in the absence of human data, data from self-play can suffice. If so, then perhaps an iterated process of generating data and then building a new state evaluator to use for both bidding and inference can result in increasingly better performance. There are some potential pitfalls to this approach; as we noted when discussing the use of UCT for bidding in Chapter 8, most hands are terrible for most game types. Therefore, we risk building a predictor that decides that the surest bet is to simply predict losses for all hands. However, if we can overcome such problems, eliminating the need for human data would significantly increase the generality of our approach to Kermit's static state evaluation.

9.2.4 Generalizing Synthetic Game Trees

In Chapter 5, we used synthetic trees to search for elementary game tree properties that might explain the success or failure of algorithms such as PIMC search. Our game tree model was, however, exceedingly simple, and could be enriched by lifting restrictions on branching factor, on uniformity of tree depth and by allowing a broader range of payoffs to name but a few examples. This generalization would also admit more complex tree properties which might help to further explain the performance of game-playing algorithms. On that note, our synthetic trees could be used to evaluate other algorithms beyond PIMC search, for example UCT with a random playout policy. Such work could lead to a general classification of different techniques and the structural environments in which they can be successfully applied.

9.2.5 Non-adversarial Domains

Throughout our work on skat, we have focused on the competitive playing strength of our player (perhaps this is an indictment of the character of the researchers involved). However, there are other applications in the domain of imperfect information games that are not so confrontational. We mentioned some of these in Chapter 3; for instance, Yan [49] discusses the problem of detecting

collusion in online bridge. This is certainly a potential issue in skat as well. Our PIPMA method from Chapter 7 could provide a starting point for tackling this problem; for instance, if a player is detected to be playing with an impossibly low mistake rate, this is strong evidence that information is somehow being obtained through external channels. A different application is that when humans play games recreationally, they do not always enjoy playing against an impossible-to-beat opponent. Although the synthetic players we describe in Chapter 7 were intended for evaluation of our computer player, a similar approach could be used for creating playing partners for humans with a highly scalable playing strength.

9.3 Conclusion

In this document, we have presented the game of skat as a context in which to study the problems posed by imperfect information games. Our major contributions consist of the creation the world's strongest computer skat player that plays on a level with human experts, a framework for explaining the success of the Perfect Information Monte Carlo (PIMC) search algorithm and assessing its a priori suitability for other game domains, and the development of a simple but effective real-time opponent modelling technique. We have also explicitly discussed the problem of inference and its role in facilitating the use of local methods such as PIMC search.

Ultimately, we hope that a greater understanding of imperfect information games will remove a significant barrier in lifting artificial intelligence techniques from games to more complex real-world domains. But we also believe that games themselves are important social and cultural artifacts. What's more, they make for a visible challenge problem that non-scientists can understand and appreciate. For these reasons, we suspect that imperfect information games such as skat will remain a fruitful area of artificial intelligence research for the foreseeable future.

Bibliography

[1] A. Amit and S. Markovitch. Learning to bid in bridge. *Machine Learning*, 63(3):287–327, 2006.

[2] D. Billings, A. Davidson, J. Schaeffer, and D. Szafron. The Challenge of Poker. *Artificial Intelligence*, 134(1-2):201–240, 2002.

[3] M. Buro. From simple features to sophisticated evaluation functions. In H. van den Herik and H. Iida, editors, *Computers and Games, Proceedings of CG98, LNCS 1558*, pages 126–145. Springer Verlag, 1999.

[4] M. Buro, J. Long, T. Furtak, and N. R. Sturtevant. Improving state evaluation, inference, and search in trick-based card games. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI2009)*, 2009.

[5] P. Ciancarini and G. Favini. Representing kriegspiel states with metapositions. In *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence (IJCAI-07)*, pages 2450–2455.

[6] P. Ciancarini and G. Favini. Monte carlo tree search techniques in the game of kriegspiel. In *Proceedings of the Twenty-First International Joint Conference on Artificial Intelligence (IJCAI-09)*, pages 474–479, 2009.

[7] A. Davidson, D. Billings, J. Schaeffer, and D. Szafron. Improved opponent modeling in poker. In *Proceedings of the 2000 International Conference on Artificial Intelligence*, pages 1467–1473. Citeseer, 2000.

[8] H. Finnsson and Y. Bjornsson. Simulation-based approach to general game playing. In *Proceedings of AAAI '08*, pages 259–264, 2008.

[9] I. Frank and D. Basin. Search in games with incomplete information: A case study using bridge card play. *Artificial Intelligence*, pages 87–123, 1998.

[10] I. Frank, D. Basin, and A. Bundy. Combining knowledge and search to solve single-suit bridge. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence*, pages 195–200, 2000.

[11] T. Furtak and M. Buro. Minimum proof graphs and fastest-cut-first search heuristics. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI2009)*, 2009.

[12] T. Furtak and M. Buro. Using payoff-similarity to speed up search. In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI2011)*, 2011.

[13] S. Ganzfried and T. Sandholm. Game theory-based opponent modeling in large imperfect-information games. In *Proceedings of the Tenth International Conference on Autonomous Agents and Mutli-Agent Systems (AAMAS)*, 2011.

[14] A. Gilpin and T. Sandholm. Finding equilibria in large sequential games of imperfect information. In *ACM Conference on Electronic Commerce*, 2006.

[15] M. Ginsberg. GIB: Imperfect Information in a Computationally Challenging Game. *Journal of Artificial Intelligence Research*, pages 303–358, 2001.

[16] M. Johanson. Robust Strategies and Counter-Strategies: Building a Champion Level Computer Poker Player. Master's thesis, University of Alberta, Canada, 2007.

[17] M. Johanson and M. Bowling. Data biased robust counter strategies. In *Proceedings of the Twelfth International Conference on Artificial Intelligence and Statistics (AISTATS)*, pages 264–271, 2009.

[18] M. Johanson, M. Zinkevich, and M. Bowling. Computing Robust Counter-Strategies. In *Advances in Neural Information Processing Systems*, 2008.

[19] T. Keller and S. Kupferschmid. Automatic Bidding for the Game of Skat. In *Proceedings of the 31st Annual German Conference on AI (KI 2008)*. Springer-Verlag, 2008.

[20] R. Kemp, B. McKenzie, and E. Kemp. Issues in Designing Tutors for Games of Incomplete Information: a Bridge Case Study. *Research and Development in Intelligent Systems XXII*, pages 331–344, 2006.

[21] L. Kocsis and C. Szepesvari. Bandit Based Monte-Carlo Planning. In *Proceedings of the European Conference on Machine Learning*, pages 282–293, 2006.

[22] D. Koller, N. Megiddo, and B. Von Stengel. Fast algorithms for finding randomized strategies in game trees. In *Proceedings of the twenty-sixth annual ACM symposium on Theory of computing*, pages 750–759. ACM, 1994.

[23] S. Kupferschmid and M. Helmert. A Skat Player based on Monte-Carlo Simulation. In *Proceedings of the 5th International Conference on Computers and Games*, pages 135–147. Springer-Verlag, 2007.

[24] C. Lemke and J. Howson. Equilibrium points of bimatrix games. *Journal of the Society for Industrial and Applied Mathematics*, 12(2):413–423, 1964.

[25] D. Levy. *The million pound bridge program*. Ellis Horwood, Asilomar, CA, 1989.

[26] J. Long, N. R. Sturtevant, M. Buro, and T. Furtak. Understanding the success of perfect information monte carlo sampling in game tree search. In *Proceedings of the 24th AAAI Conference on Artificial Intelligence (AAAI2010)*, 2010.

[27] J. R. Long and M. Buro. Real-time opponent modelling in trick-taking card games. In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI2011)*, page To appear, 2011.

[28] J. Méhat and T. Cazenave. Ary, a general game playing program. In *13th Board Game Studies Colloquium*, 2010.

[29] K. Mossakowski and J. Mandziuk. Learning without human expertise: a case study of the double dummy bridge problem. *Neural Networks, IEEE Transactions on*, 20(2):278–299, 2009.

[30] J. Nash. Equilibrium points in n-person games. *Proceedings of the National Academy of Sciences of the United States of America*, 36(1):48–49, 1950.

[31] A. Parker, D. Nau, and V. Subrahmanian. Game-tree search with combinatorially large belief states. In *International Joint Conference on Artificial Intelligence*, volume 19, page 254. Citeseer, 2005.

[32] M. Richards and E. Amir. Opponent modeling in scrabble. In M. M. Veloso, editor, *IJCAI*, pages 1482–1487, 2007.

[33] N. Risk and D. Szafron. Using counterfactual regret minimization to create competitive multiplayer poker agents. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: volume 1-Volume 1*, pages 159–166. International Foundation for Autonomous Agents and Multiagent Systems, 2010.

[34] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, Englewood Cliffs, NJ, 2nd edition, 2003.

[35] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, Englewood Cliffs, NJ, 3rd edition, 2009.

[36] A. Samuel. Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development*, 3(3):211–229, 1959.

[37] J. Schaeffer, R. Lake, P. Lu, and M. Bryant. Chinook: The world man-machine checkers champion. *AI Magazine*, 17(1):21–29, 1996.

[38] J. Schäfer. The UCT Algorithm Applied to Games with Imperfect Information. Master's thesis, University of Magdeburg, Germany, October 2007.

[39] C. Shannon. A chess-playing machine. *Scientific American*, pages 48–51, 1950.

[40] B. Sheppard. Mastering scrabble. *IEEE Intelligent Systems*, 14(6):15–16, 1999.

[41] S. Smith, D. Nau, and T. Throop. Computer bridge: A big win for ai planning. *AI Magazine*, 19(2):93, 1998.

[42] F. Southey, M. Bowling, B. Larson, C. Piccione, N. Burch, D. Billings, and C. Rayner. Bayes' bluff: Opponent modelling in poker. In *Proceedings of the Twenty-First Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 550–558, 2005.

[43] N. Sturtevant. Current Challenges in Multi-Player Game Search. In *Proceedings of the 4th International Conference on Computers and Games*, 2004.

[44] N. Sturtevant. An Analysis of UCT in Multi-Player Games. In *Computers and Games*, 2008.

[45] N. Sturtevant and A. White. Feature Construction for Reinforcement Learning in Hearts. In *Proceedings of the 5th International Conference on Computers and Games*, 2006.

[46] G. Tesauro. TD-Gammon, a self-teaching backgammon program, reaches master-level play. *Neural Computation*, 6(2):215–219, 1994.

[47] B. von Stengel and D. Koller. Team-Maxmin Equilibria. *Games and Economic Behavior*, pages 309–321, 1997.

[48] Y. Wang and S. Gelly. Modifications of UCT and sequence-like simulations for Monte Carlo Go. In *Proceedings of the 2007 IEEE Symposium on Computational Intelligence in Games*, pages 175–182, 2007.

[49] J. Yan. Collusion Detection in Online Bridge. In *Twenty-Fourth AAAI Conference on Artificial Intelligence*, 2010.

[50] M. Zinkevich, M. Johanson, M. Bowling, and C. Piccione. Regret minimization in games with incomplete information. *Advances in Neural Information Processing Systems*, 20:1729–1736, 2008.