Improving State Evaluation, Inference, and Search in Trick-Based Card Games

Michael Buro, Jeffrey R. Long, Timothy Furtak, and Nathan Sturtevant
Department of Computing Science, University of Alberta, Edmonton, Canada.
email: {mburo,jlong1,furtak,nathanst}@cs.ualberta.ca

Abstract

Skat is Germany's national card game played by millions of players around the world. In this paper, we present the world's first computer skat player that plays at the level of human experts. This performance is achieved by improving state evaluations using game data produced by human players and by using these state evaluations to perform *inference* on the unobserved hands of opposing players. Our results demonstrate the gains from adding inference to an imperfect information game player and show that training on data from average human players can result in expert-level playing strength.

1 Introduction

Imperfect information games pose two distinct problems: move selection and inference. Move selection is the problem of determining a good move in spite of the absence of complete knowledge concerning the state of the game, and is typically achieved through use of a search algorithm in possible combination with some form of static state evaluation. Inference is the problem of inferring hidden information in the game based on the actions of other players. These problems are inter-related; accurate inference of the state of the game should lead to more informed move selection, and good move selection should aid considerably in deciphering the information conveyed by opponent moves.

In this paper, we describe the two main techniques used to create an expert-level computer player for the game of skat. The first technique is for bidding, and involves building a static state evaluator with data from human games that overcomes some of the deficiencies of standard imperfect information search methods. The second technique is the use of inference to bias the selection of hypothetical worlds used in the imperfect information search during play.

The remainder of this paper is organized as follows. Section 2 presents an overview of skat. Section 3 describes the construction of our state evaluator, while Section 4 presents our techniques used for search and inference. Section 5 lists experimental results, and Section 6 concludes the paper.

2 The Game of Skat

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 Players Association (`www.ispaworld.org`). Although less prevalent than contract bridge in North America, there are nevertheless skat clubs in many major cities around the globe.

Skat is a trick-taking card game for 3 players. It uses a short 32-card playing deck, similar to the standard 52-card deck except cards with rank 2 through 6 have been removed. A hand begins with each of the 3 players being dealt 10 cards, with the remaining 2 cards (the *skat*) dealt face-down.

The play of the hand consists of 2 phases: bidding and cardplay. In the bidding phase, 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 soloist will then usually pick up the skat and discard any two cards face-down back to the table, although there is an option to play *hand* and not pick up the skat. The soloist then announces the *game type*, which will determine the trump suit and how many points the soloist will earn if she wins the game. There is one game type for each of the four suits (♢♡♠♣), in which the named suit and the four jacks form the trump suit. These four types are referred to as *suit games*. Other game types include *grand*, in which *only* the 4 jacks are trump, and *null*, in which there is no trump and the soloist attempts to lose every trick.

Once the soloist announces the game type, cardplay begins. This phase consists of 10 tricks, which are played in a manner similar to bridge and other trick-taking card games. The soloist's objective is to take 61 or more of the 120 available *card points*. Each card in the game is worth a fixed amount: aces are worth 11 points and 10s are worth 10. Kings, Queens and Jacks are worth 4, 3 and 2 points respectively.

We have omitted many of skat's more detailed rules from this summary; for a more thorough treatment, the interested reader is referred to `www.pagat.com/schafk/skat.html`.

From an inference perspective, one of the most interesting features of skat is that it contains hidden information not only from chance moves (i.e. the dealing out of cards), but also from player moves in the form of the soloist's 2-card discard. This property is absent from many other popular card games,

such as bridge, blackjack and poker. Skat can also be described as a benign form of a multi-player game. The bidding phase is competitive between all three players, but, once card play begins, the game can somewhat safely be abstracted as a two-player game. Nevertheless, there is a need for players to infer the cards of both their opponent and their partner, creating opportunities for signaling and information sharing that would not exist in a two-player game.

3 Improving State Evaluation

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 [Tesauro, 1994]. 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, 2006] describes 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.

We propose approximating imperfect information state values directly and using such estimates in game tree search. The obvious benefit over perfect-information-based evaluations is that programs can judge state merits more accurately, taking players' ignorance into account. In skat, for instance, open-handed game values of null-games severely underestimate the soloist's winning chance.

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 which, when combined with search methods such as Monte Carlo, approximate optimal policies.

3.1 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 [Buro, 1999], 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

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

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.

3.2 Application to Skat Bidding

One big weakness of today's skat programs is bidding. Bidding is commonly implemented using rule-based systems designed by program authors who may not be game experts. Kupferschmid et al. approach the problem in more principled ways. Kupferschmid and Helmert [2007] discuss learning a linear success predictor from open-hand game (often abbreviated as *DDS*, which stands for *double dummy solver*) values given some features of the soloist's hand and the skat. Unfortunately, no performance figures were presented. In a follow-up paper, Keller and Kupferschmid [2008] describe 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. Their experimental results indicate that the system is able to identify winnable hands with confidence, but no tournament results have been reported.

Our bidding system is also 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"), but instead of using a linear model we estimate winning probabilities by means of logistic regression. Basing the strength of hands on winning probability or expected payoff rather than expected card points 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 it is likely that players of a wide range of skill are represented in the data.

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 10-card hand h in playing position $p \in \{0, 1, 2\}$ having discarded skat s.

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 10 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 10s 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 10 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 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. $\clubsuit^\star(\clubsuit 78 \mathrm{J} \heartsuit \mathrm{JA}) = 2^0 + 2^1 = 3$.
- $\{\overline{\clubsuit^\star}, \overline{\spadesuit^\star}, \overline{\heartsuit^\star}, \overline{\diamondsuit^\star}\}(c) \in \{0, \ldots, 2^{11}-1\}$ is the same as $\{\clubsuit^\star, \spadesuit^\star, \heartsuit^\star, \diamondsuit^\star\}(c)$, but includes all jacks. E.g. $\overline{\clubsuit^\star}(\clubsuit 78 \mathrm{J} \heartsuit \mathrm{JA}) = 2^0 + 2^1 + 2^{10} + 2^8 = 1283$.
- $\mathrm{tc}(t,c) \in \{0,1\}$ 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 ♣ is trump):

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

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

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

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

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 1. The root value is the result of an expecti-max computation of winning probability over all soloist choices (picking up the skat or not, selecting the game type (6 choices) to

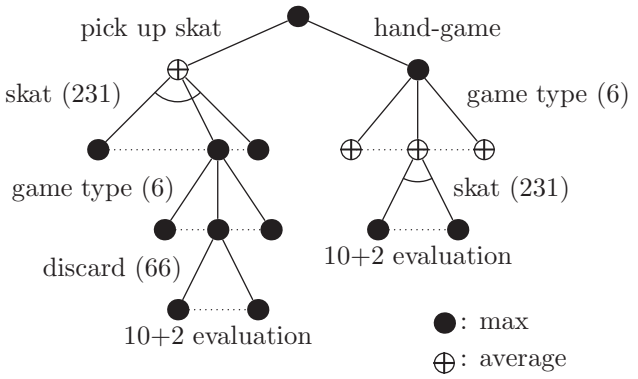

Figure 1: The simplified skat bidding search tree based on 10+2 evaluation.

be played, and discarding two cards (66 choices)) as well as chance nodes for the initial skat (22 choose $2 = 231$ possibilities).

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 our program 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, the program receives the skat cards and selects the contract and the cards to discard based on searching the relevant subtree.

4 Improving Search & Inference During Play

The Perfect Information Monte Carlo (PIMC) search algorithm is one of the most widely used methods for dealing with imperfect information games, and card games in particular. The algorithm works as follows. At the root node of the search, we first create a hypothetical world, w, where we assign a value to all unobserved variables in the current game state. In a card game, this would consist of distributing cards between the unobserved hands of our opponents. We then assume that all players have perfect information of the game and can therefore use standard perfect information game search methods, such as minimax with alpha-beta pruning, to determine the best move at the root for that world w. We record which move was best and then repeat this process as many times as possible, choosing a different w for each iteration. In the end, we simply play the move at the root which was most frequently recorded as the best move.

The application of PIMC search to card games, and in particular to the game of contract bridge, was first proposed by Levy [1989], and successfully implemented by Ginsberg [2001]. PIMC has recently been applied to skat by Kupferschmid and Helmert [2007] and to hearts and spades by Sturtevant and White [2006]. It is also the search algorithm we use for the skat program presented in this paper.

More recently, the UCT algorithm by Kocsis and Szepesvari [2006] has emerged as a candidate search algorithm for imperfect information games. UCT differs from Monte Carlo search in that statistics on moves are kept at interior nodes of the game tree instead of only at the root, and at each pass (or simulation) through this tree, moves are selected according to these statistics. UCT has been implemented in skat by Schäfer [2007] and in hearts by Sturtevant [2008]. Although in skat, UCT was unable to defeat PIMC search, in hearts it proved to be stronger than the previous best known computer hearts players.

4.1 Perfect Information Search Enhancements

When evaluating moves in a perfect information setting, our program 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 [Furtak and Buro, 2009], 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 [Kupferschmid and Helmert, 2007], we group cards according to their strength and only search one representative move in each group. For instance, when holding 7 8 suited in ones 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. Finally, we added node-depth information to null-game leaf evaluations which causes our program to delay losses as soloist and speed-up wins when playing as defender.

4.2 Previous Work on Game State 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 man-

ner to what we describe here includes work by Ginsberg in bridge [2001], and Richards and Amir in Scrabble [2007]. 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 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 runtime. 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 express the probability of a hypothetical world as follows:

$$P(\text{world} \mid \text{move}) = \prod_i P(f_i) P(f_i \mid \text{move}) \qquad (1)$$

Here, $f_1 \ldots f_n$ are all of the features present in the hypothetical world. $P(f_i \mid \text{move})$ 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.

4.4 Application to Skat Cardplay

The current version of our skat program draws conclusions from bids and the contract based on (1), 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 10-card hands w.r.t. bids we could in principle make use of the bidding system we described in Subsection 3.2. 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, such as suit-length and high-card distributions, to quickly approximate hand probabilities by multiplying feature value frequencies that have been estimated from generated bidding data.

The same basic algorithm is used for defenders, 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. Again, this is done using our 10+2 evaluation. In addition, as a defender our program also considers all possible discard options for the soloist.

5 Experiments

In this section we present performance results for our skat program, *Kermit*, that indicate that it is 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.

5.1 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 1 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 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.* [2007; 2008]. 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. 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. We suspect a bug in KNNDDSS's implementation and are working with its authors to resolve this issue.

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

Table 1: 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.

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

Table 2: 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.

5.2 Playing on a Skat Server

To gauge Kermit's playing strength compared to that of good human players, we gathered game data from a skat server, which we summarize in Table 2. The top entries list the results of Zoot, Kermit, XSkat, and Bernie playing against Zoot and Kermit, where Zoot is a clone of Kermit using bidding threshold 0.55 instead of 0.6. Bernie is a program written by Jan Schäfer [2007] that uses UCT search with XSkat as playout module in all game phases. The results show that the Kermit variations dominate these two programs when averaging over all games they played together 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 indicate a statistically significant

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

6 Conclusion and Future Work

In this paper, 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 outperforms other existing bidding methods.

When constructing Kermit, we were fortunate to have access to a large volume of human game data. In future work, we would like to answer the question of whether, in the absence of human data, synthetic data from self-play can suffice. If so, then perhaps an iterated process of generating data and then building a new state evaluator can result in increasingly better performance. Another avenue for improvement is to modify the Monte Carlo search procedure so as to take opponent ignorance into account. This is particularly important in skat, since null-games are nearly always a loss under the perfect information assumption made by double dummy solvers yet are often winnable in practice. Finally, at the time of this writing we have started to experiment with inference based on card-play which we believe is essential for reaching world-class playing strength.

Acknowledgments

We would like to express our gratitude to everyone who helped in making this project a success. Ron Link's expert advice on skat hand evaluation and the large number of human games made available to us by an anonymous skat server administrator have been instrumental in contructing Kermit's evaluation function. Frequent discussions with Jan Schäfer and Karen Buro helped us sorting out good from bad ideas for improving our program, and without the world-caliber skat players Ron Link, Eric Luz, and Rainer Gößl playing thousands of games against Kermit we wouldn't have been able to conclude that our program has reached expert playing strength. We also thank Gunter Gerhardt, Sebastian Kupferschmid, and Thomas Keller for access to their skat programs and their help in resolving some implementation issues. Financial support was provided by NSERC and iCORE.

References

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

[Furtak and Buro, 2009] 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.

[Ginsberg, 2001] M. Ginsberg. GIB: Imperfect information in a computationally challenging game. *Journal of Artificial Intelligence Research*, pages 303–358, 2001.

[Keller and Kupferschmid, 2008] T. Keller and S. Kupferschmid. Automatic bidding for the game of skat. In *31st Annual German Conference on AI (KI 2008)*. Springer-Verlag, 2008.

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

[Kupferschmid and Helmert, 2007] 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.

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

[Richards and Amir, 2007] M. Richards and E. Amir. Opponent modeling in Scrabble. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI2007)*, 2007.

[Schäfer, 2007] J. Schäfer. The UCT algorithm applied to games with imperfect information. Master's thesis, University of Magdeburg, Germany, October 2007.

[Sturtevant and White, 2006] N. Sturtevant and A.M. White. Feature construction for reinforcement learning in hearts. In *Computers and Games*, pages 122–134, 2006.

[Sturtevant, 2008] N. Sturtevant. An analysis of UCT in multi-player games. In *Computers and Games*, 2008.

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