Proceedings of the Twelfth International
Symposium on Combinatorial Search (SoCS 2019)

Challenging Human Supremacy in Skat

Stefan Edelkamp
King's College London, UK
stefan.edelkamp@kcl.ac.uk

Abstract

After impressive successes in deterministic and fully-observable board games to significantly outclass humans, game playing research shifts towards non-deterministic and imperfect information card games, where humans are still persistently better. In this paper we devise a player that challenges human supremacy in Skat. We provide a complete player for playing selected variants of the game, with effective solutions for bidding and Skat putting, eliciting knowledge extracted from several million games. For trick play we combine expert rules with engineered tree exploration for optimal open card play. For dealing with uncertainty especially in Ouvert games we search the belief space.

Introduction

Many known fully-observable deterministic combinatorial games are either solved (Gasser 1995; Schaeffer et al. 2005; Allis 1998) or computers significantly outperform human play (Campbell, A. J. Hoane, and Hsu 2002). One recent progress in Go exploited the prediction of the next move with a deep neural network that was trained on a wider selection of expert games (Silver and et al. 2016). Deep Mind's *AlphaZero*, which takes the rules of the games and, then, applies reinforcement learning mainly via self-playing millions of games, has reached world-class play both in Shogi, and Chess (Silver et al. 2017). In contrast, card games with randomness (in the deal) and imperfect information (due to hidden cards) are still a major challenge to AI game playing technology. One recent exception is Heads-Up Limit (Texas) Hold'em Poker (Bowling et al. 2017).

For trick-based card games, however, the situation is unsatisfactory. After early advances in playing Bridge (Ginsberg 1999), research progress has slowed down considerably, even though computer programs are under continuous development. In Skat (Grandmontagne 2005; Harmel 2016; Lasker 1938; Kinback 2007; Quambusch 1990; Schettler and Kirsch bach 1988; Schubert 1887; Wergin 1975), an international three-player card game using a deck of 32 cards, with a deal of 10 cards for each player and 2 left-overs forming the so-called *skat*, there is more recent research activity. Work on an efficient open card solver (Kupferschmid and Helmert 2006) went into an expert-level Skat player (Long 2011). Symmetries and refined search algorithms have been studied by (Furtak 2013). For world-class play of Skat, however, there still is a gap. We present an AI that —in the long run— is expected to outperform world-class human play. As a first result on this research avenue, we present a player for the Nullspiel, which is particularly interesting, as existing Skat programs play weak. We also contribute initially results for playing Trump, especially Grand. We compile the information of millions of games in a clever way, and use a combination of expert rules, aggregated statistical information on winning probabilities and information gain, as well as high-performance exploration algorithms. We indicate that an AI can outperform human play.

The paper is structured as follows. First, we explain the rules of Skat. Next, we shed light on the bidding and skat putting process, exploiting expert wisdom acquired over the years in the form of a database with millions of played games. We walk through the stages and the versions of the game, which lead to different player proposals. We rely on statistical information and expert rules, engineered and novel search algorithms. The exploration is enriched with knowledge moves (e.g., to reduce the amount of uncertainty in the remaining cards). Experimental results show that our program can surpass human play.

The Game of Skat

The rules of Skat go back to Hempel around 1848. Competitive Skat is defined by the International Skat Player Association (www.ispaworld.info), The game is played with three players. A full deck has 8 cards (A, T, K, Q, J, 9, 8, and 7) in all four suits (♣, ♠, ♡, ♢). After shuffling, each player receives 10 cards, while the skat consists of two cards. There are four phases of the Skat game: the bidding stage, taking and putting the skat, and the actual play for tricks. The *declarer*, who won the *bidding*, is playing for a win against the remaining two *opponents*. She is allowed to strengthen her hand by taking the skat and putting it (can be the same ones).

For a strong Skat AI, especially for the early stages of the games like bidding, most interestingly is approximating the probability $P^t_w(h)$ of winning a given hand h in a game of type t (Gößl 2019). The games being played and bid for are *Trump*, which includes *Grand* and *Suit* (♣, ♠, ♡, or ♢), as well as *Nullspiel*, a trick-avoiding variant of the game.

Nullspiel

There are four variants of the Nullspiel: *Null* (bidding value 23), *Null Hand* (35), *Null Ouvert* (NO, 46), and *NO Hand* (59), where *Ouvert* forces the declarer to show all her cards prior to the play, and *Hand* prohibits the declarer to take the skat. The declarer that wins the bidding, must lose all tricks. In this variant according to the expert judgment (see www.skatfuchs.eu for a list of studied programs) most computer card game progams play badly.

In the Nullspiel the ordering of cards is A, K, Q, J, T, 9, 8, and 7. If the declarer gets any trick, he loses. To the contrary, she wins by certain if her hand is *safe*.

Definition 1 (Safe Card) *A declarer's card c is* safe*, if all gaps g in its suit with value lower than c are supported by at least the matching number of cards below g, with a special case for the declarer's turn in the first trick, where an extra support card is needed. The declarer's hand h is* safe *if all cards c in h are safe.*

Refined definitions of a safe hand include the cards in the skat, and accommodate the cards that have being played.

Strategies for the declarer and her opponents depend on the position of the players within the trick and dominate expert play. Playing agreements like *Shortest Suit – Smallest Card First* indicate the fundamental importance of collaboration between the two opponents for maximizing the exchange of information. Such hidden rules are difficult for an AI to learn automatically, especially given that for several of such simple rules, there are exceptions in world-class play.

There are other subtleties on knowledge elicitation given that there only two possible outcomes of the Nullspiel. One immediate consequence is that longer play often is a safer way to win the game of the opponents. In case a suit has to be obeyed, the card distribution in this suit is crucial, and, if not, dropping card strategies play an important role.

For each of the suits s in a hand h we determine the probability of winning $P_w(h, s)$, using a multinomial distribution (refined with the winning probabilities found in the expert games). The probabilities are stored in tables, and the winning probabilities $P_w(s)$ among all the suits are multiplied as an estimate for the overall winning probability $P_w(h)$.

Trump Games

Card values in Trump games are added for the tricks made and the skat put, with a usual split at 60 of the 120 possible points. Other contracts (90, 119) are possible. The bidding value depends on the distribution of Js: As the multiplier of the color value ($12 = \clubsuit$, $11 = \spadesuit$, $T = \heartsuit$, or $9 = \diamondsuit$) or 24 (Grand), 1 is added to the number of consecutive Js in the order $\clubsuit$, $\spadesuit$, $\heartsuit$, and $\diamondsuit$; or the number of consecutive Js in the joint hands of the opponents.

Definition 2 (Standing Card) *A declarer's non-trump card c is* standing *in a suit s, if c cannot be beaten by all other cards in s still being present in the game. The declarer's hand h is* standing *if all non-trump cards c in h are standing.*

We implemented a more refined definition of standing cards that depends on the distribution of the Js, the position in the trick, and that takes the possible distribution of opponent cards and the experience in expert games into account. To estimate the number of tricks possible, we, thus, introduce fractional values for standing cards and the concept of virtual standing cards that are likely to become standing cards during play.

Bidding

In case of no *jump bids*, the usual bidding stage follows a predefined order of calling numbers corresponding to the *value* of the game $(18, 20, 22, 23, 24, 27, 30, 33, 35, \ldots)$. The final bidding value indicates in which suits the players are strong, so that we maintain a table of the at most two games fitting the final bid. In opponent play this information often determines the choice of the first card to be played.

Different computer bidding strategies have been proposed; in the literature, among others we find neural networks (Kupferschmid and Helmert 2006), nearest neighbor search (Keller and Kupferschmid 2008), and statistical analyses of human games together with single-agent game tree search (Long 2011).

For computer bidding we use our rather accurate knowledge on winning probabilities $P_w^t(h)$. In contrast to classical learning and game-theoretical approaches, we exploit flat statistical knowledge on the individual strength of the cards in hand that we have extracted from millions of expert games. We estimate probabilities $P_w^t(h)$ to win a game t and determine the expected payoff of the game t (Gößl 2019), which for the bidding stage is maximized.

Given $3\binom{32}{10,10,10,2} = 8\,259\,883\,225\,513\,920$ different Skat deals (including the turn) storing a lookup table for $P_w^t(h)$ in memory clearly is infeasible and, even more importantly, for many million expert games, way too sparse to retrieve robust values for each game. Note the conceptual difference in P_w^t prior and posterior to skat taking. Before the skat is taken, the maximal bidding value is determined via computing the average payoff over all possible $\binom{22}{2} = 231$ skats. For putting we take the maximum over all $\binom{12}{2} = 66$ options to select the two cards for the skat.

The complete bidding strategy is more complex and exceeds the scope of the paper. For example, strong players that announce the bid, often drop out one step before calling the actual value, not to be forced to play in case of a tie.

Nullspiel

We follow the suggestion of Emanuel Lasker (1929) (the only German Chess world champion!), and combine statistics on winning probabilities in each suit as follows

$$P_w(h) = \prod_{s \in \{\clubsuit, \spadesuit, \heartsuit, \diamondsuit\}} P_w(h, s).$$

There are different probability tables $P_w(h, s)$ for different variants of the Nullspiel. In each suit, we have eight cards and $2^8 = 256$ selections of cards that form a pattern. As the binary representation of the position in the vector and the pattern are identical, given the bitvector for a given hand, we can extract the probability value $P_w(h, s)$ in time $O(1)$ by bit masking, bit shifting and bit reversal operations.

Trump Games

For Trump games we used hash functions to store similar hands in a smaller table (with several 10K entries). Such functions can be thought of selecting a set of features that generate equivalence classes of hands.

For example, in Grand games we identified the following winning features to be sufficient (in some cases, like bidding values and points put in skat, we cluster the set of possible values into a smaller set):

1. value put in the skat, we group the values into 5 classes;
2. trump quality encoding distribution in J: 10 classes;
3. number of As and Ts, as two independent features;
4. estimated number of lost cards based on standing ones.
5. position on table (who's turn it is);
6. bidding value, we group the values into 4 classes

For Suit games, we identified 9 features: trump count, number of trump high cards (As+Ts), encoding of Js, non-trump As and Ts, missing suits, position of player, encoded skat and bidding value. If we denote $f_1, \ldots, f_l$ for the l features, based on expert games we built a probability table

$$P_w(h) = hash(f_1(h), \ldots, f_l(h)).$$

Statistical evidence has been collected that the chosen features are indeed relevant (Gößl 2019). Choosing $P_w(h)$ for weakly supported feature combinations is a learning problem, which we resolve via returning a neighbor entry. We also tried to learn $P_w(h)$ with multivariable linear regression, but the predictions were often off by 20% and more.

Skat Putting

Once the skat has been taken by the declarer, there are $\binom{12}{2} = 66$ options for putting it. We derived a strategy to select a skat that optimizes the *dropping gain*.

Definition 3 (Dropping Gain) *The* dropping gain *is the change in the winning probability when removing a card from the hand. If h is the hand before the drop and h' is reduced hand after the card drop in suit s, then we have*

$$drop(h, h') = P_w(h, s) - P_w(h', s).$$

In other words, for the choice of skat cards, we prefer the ones that improve the winning probability the most.

While for Ouvert games this is the default, for closed games we allow exceptions: because of dropping opportunities, we may afford to keep a higher-risk single card in one suit to favor more secure suits.

As a refinement for the skat putting strategy described above, we keep three different tables depending on whether 0, 1, or 2 cards of a given suit are put into the skat.

Nullspiel

We validated that the deviation of the predicted winning probabilities in this bidding and skat putting strategies compared to the expert play in the large set of 6M games was at most 3% (see Figure 5). For us this observation came as a surprise as the crude approximation of winning chances by multiplying the winning probabilities in each suit fully neglects card dropping, which is crucial for playing the game. We, therefore, decided to go in for this option for the declarer to select the cards to be put and dropped.

Trump Games

There are many special rules to detect games won by certain. One general rule in Grand is the following.

Theorem 1 (High-Card Theorem) *If the number of high-value cards (HC) secured by the declarer is at least as large as the number of tricks lost (assuming no points were put into them), then the standard Grand game is won.*

Proof: We look at the following cases.

1 Cached HC For one opponent trick, they certainly cannot reach 60 points.

2 Cached HCs For 2 tricks at most 44 points are available (4 As).

3 Cached HCs In 3 opponent tricks at most 58 points can be made (4 As, 1 T, and 1 K).

4 Cached HCs In 4 tricks at most 59 points are possible (3 As, 1 T, and 4 Ks).

5 Cached HCs In 5 lost tricks at most 57 points are contained (2 As, 1 T, 4 Ks and 3 Qs)

6 Cached Hs In 6 opponent tricks at most 54 points are possible (2 As, 1 T, 4 Ks, 3 Qs, and 2 Js)

7 Cached HCs In 7 opponent tricks we find at most 47 points (1 A, 4 Ks, 4 Qs and 4 Js)

If one has 4 As, then for the case of 4 cached HCs the declares can even afford giving 1 Q to the opponents, offering the opponents again at most 59 points in total.

The next concept to be understood is the concept of *standing cards*. Roughly speaking a standing card is a (trump or non-trump) card, which will go home by certain. While this definition suggests an integer value, we refine the value by attaching probabilities for each card to be saved. This takes the e.g,, the current turn, and the values put in the skat into account. We use a table for storing these values, which were derived on the basis on expert knowledge assisted by the statistical analysis of several million of played Grand games.

In each suit, we have seven non-trump cards and $2^7 = 128$ cards of that suit that form a pattern.

Open Game Play

For open games we implemented an engineered *glass box* solver to decide a game; the game-theoretical value for trump games is then found via binary search.

To represent hands, the skat and the played cards we employ bitvectors (in form of unsigned integers), for which we utilize Boolean set operations: `&` , `~` , and `|` . so that bit masking and shifts help to identify chosen parts of the hands in constant time. For an efficient solver, we exploit that modern CPUs provide constant-time `__builtin` procedures to determine the number of cards as `POPCOUNT(x)` (short `|x|`), the first card as `LZCOUNT(x)`, and the last card as `TZCOUNT(x)` (short `select`). For the representation of a *state* of the game, we chose the union of the three hands.

```
AND()
  if (v = lookup(hand[0]|hand[1]|hand[2],0)) return v;
  h = hand[0];
  while (h)
    index = select(h);
    bitindex = (1<<index);
    if (!playable(hand[0],index,0) || equiv(h,index))
      h &= ~bitindex;
      continue;
    hand[0] &= ~bitindex; played |= bitindex;
    t[0] = i[0]; t[1] = i[1]; t[2] = i[2]; i[0] = index;
    if (|played| % 3 == 0)
      turn = winner(0);
      i[0] = i[1] = i[2] = -1;
      if (turn == 0) val = 0;
      else if ((hand[0]|hand[1]|hand[2]) == 0) val = 1;
      else if (safe(hand[0],played) == hand[0]) val = 1;
      else if (turn == 1) val = OR1();
      else if (turn == 2) val = OR2();
    else val = OR1();
    i[1] = t[1]; i[2] = t[2]; i[0] = t[0];
    played &= ~bitindex; hand[0] |= bitindex;
    h &= ~bitindex;
    if (|played| % 3 == 0)
      add(hand[0]|hand[1]|hand[2],0,1);
      return 1;
  if (|played| % 3 == 0)
    add(hand[0]|hand[1]|hand[2],0,0);
  return 0;
```

Figure 1: Glass box for the declarer's turn in the Nullspiel.

Nullspiel

Fig. 1 provides the glass box pseudo-code implementation including transposition table pruning, detection of equivalent cards, and analyses of safe cards[1]. A concise implementation of the decision procedure to compute safe cards is given in Fig. 2. The backtracking algorithm in Fig. 1 returns the game-theoretical value (lost, won) of the game at a given node in the search tree, with an And-node (AND) referring to the declarer and an Or-node referring to one of the two opponents (OR1 and OR2). Code fragments for the latter twos are similar, with the outcomes 0 and 1 reversed.

We see a lot of bitshifting to convert an index of a card to its position in the bitvector and vice versa. All variables not bound are global and set in a main driver program of the search. Variables set to some value are set back to their original one on a backtrack. We check for early termination in case the player only has safe cards. A transposition table (a hash table supporting insertion and membership only) avoids evaluating same game states again. We have two functions that check a card from a given hand h (maintained as a bitvector) is playable according to the rules of the game and not equivalent, meaning that a smaller card exists that will lead to the same game value, because these cards are adjacent. The recursive structure generates a tree and using

[1]We tried *proof-number search* (Allis, van der Meulen, and van den Herik 1994), but by the involved handling of transpositions and the experienced higher efforts per node, it was less efficient.

```
safe(hand, played)
  s = 0;
  for (suit=0;suit<4;suit++)
    counter = 0;
    for (j=7;j>=0;j--)
      card = 1 << (suit*8+j);
      if (card & played) continue;
      if (hand & card) counter++; s |= card;
      else if (counter == 0) break; counter--;
  return s;
```

Figure 2: Computing safe cards in the hand of the declarer with respect to a set of already played cards.

Boolean reasoning to generate an optimal strategy. Pruning takes place in the Boolean formulas as only parts of it (e.g., the principal variation) need to be evaluated.

If there are three cards are on the table, the trick is evaluated and the game is either terminated, or continued with the winner of the trick. To avoid problems with ongoing tricks, we use the transposition table only after a trick has been played. This is sufficient to encode the entire state into 32 bits, re-using the skat cards that during play of a given game remain unchanged to denote the turn.

Trump Games

Fig. 3 shows the according glass box for Trump games, where point scores for both parties are added and hashed. To maintain the card order within the tricks, which is important to evaluate its outcome, we use explicit indices.

Playing Null

The algorithm for hidden card games is significantly different to Ouvert games, for which most of the cards (except of the ones in the skat) are known to the opponent players. As indicated above, in both cases, not the shortest but the safest way to win is sought. We advice not to give up, if a game lost for the opponents, since the declarer may not be aware of this fact and, thus, his play can be flawed.

There are 177 (of the 256) patterns of unsafe cards in a suit, if it is not the declarer's turn, and 209 patterns, if it is the declarer's turn (as the forced play of a card in a safe group may cast it unsafe). Safe patterns have a winning probability of 100% in our look-up table. The algorithm, consisting of a set of expert rules, roughly works as follows.

For the choice of the first card of the declarer we again use a table addressed with patterns. The first table gives the probability of winning by choosing this card, the second one provided the index of the card itself. This information is elicited from human play. The probabilities are refined on whether or not one or two cards of the same color have been put.

Opponents' Choice

Choice of Initial Card If there is a 7 on your hand, choose the shortest suit, of which you do not have another 7. If the 7 is sole, play it immediately. If there are two or more short colors, choose the one with the higher winning probability. For example in ($\clubsuit K$, $\clubsuit Q$, $\spadesuit J$, and $\spadesuit 8$) choose

```
AND(remaining)
  best = lookup(hands[0]|hands[1]|hands[2],as);
  if (best != -1) return best;
  while (remaining) {
    index = select(remaining);
    bitindex = (1<<index);
    if (!playable(hands[0],index,0))
      remainng &= ~bitindex; continue;
    hands[0] &= ~bitindex; played |= bitindex;
    i[0] = index; val = -1;
    if (|played| % 3 == 0)
      cached = played; turn = winner(1,2,0);
      score = VALUE(i[0]) + VALUE(i[1]) + VALUE(i[2]);
      if (turn) gs += score; else as += score;
      t1 = i[1], t2 = i[2]; i[0] = i[1] = i[2] = -1;
      val =
        (gs >= 120-LIMIT) ? 0 : (as > LIMIT) ? 1 :
        (turn == 0) ? AND(hands[0]) :
        (turn == 1) ? OR1(hands[1]) : OR2(hands[2]);
      i[1] = t1; i[2] = t2;
      if (turn) gs -=score; else as -= score;
      cached = played;
    else
      val = OR1(playable(hands[1],(i[2]>=0)?i[2]:i[0]));
    i[0] = -1;
    played &= ~bitindex; hands[0] |= bitindex;
    remaining &= ~bitindex;
    if (val == 1)
      if (|played| % 3 == 0)
        add(hands[0]|hands[1]|hands[2],as+128);
      return 1;
  if (|played| % 3 == 0)
    add(hands[0]|hands[1]|hands[2],as);
  return 0;
```

Figure 3: Glass box for the declarer's turn in Trump games.

spades, and take the smallest card in there. In contrast, if you have more than one 7 on your hand, take the suit, of which you do not have the 7, choose the longest, and play the highest card.

Reacting on Cards If the other opponent has all higher cards of the played suit and another lower card, on which the declarer can be beaten, take the trick and play the lower card immediately. For example for $\diamondsuit K$ being played and $(\diamondsuit A, \diamondsuit Q, \diamondsuit 7)$ in hand, play $\diamondsuit A$ and then play $\diamondsuit 7$, not $\diamondsuit Q$. To generate dropping cards this scheme also applies if the declarer cannot be beaten: the suit, on which the other opponent drops cards is then to be played, as long as there is the chance to beat the declarer. If one has all remaining cards in one suit and also the 7, then the 7 is played to show that the color should not be played further on. Be careful to assume that the declarer has all cards on hand, as she may have put some into the skat.

Change of Suit If one does not have any suit with small cards, a change of suits is appropriate. More precisely, a change in suit is necessarily needed, if: a) the declarer has no more card in this played suit (an exception of this rule is if the other opponent has already dropped a card in a suit, from which one has the smallest, and the declarer cannot get rid of his weaknesses); b) There is no suit, which threatens the declarer (such as 7, or 8, or 8 and 9, or 8 and T, or 9 and T, or 9 and T and J, or 9 and T and Q); c) the other opponent can't drop all cards on the played suit; d) one has a singleton that can be dropped.

Dropping Card A general rule is to play the highest card of the shortest color of which one doesn't has a 7 or a once-supported 8. Do not drop the card that is sole in supporting the 8. In case of two suits of same length, the suit is chosen, of which one has the highest lowest card.

Declarer's Choice

For the declarer, we distinguish between selecting the first card (only for first trick) and reacting to cards on the table.

Declarer's Card Choose the suit which has the highest winning probability according to the probability distribution table. This might be a sole 8 or, in case of 8 and 9, also the 8 (playing 9, is of course, equivalent).

Obeying a Suit Here we assume a card in hand for the offered suit. If there is only one card on the table, the card is selected that has a value right below the one that is on the table. If two cards have been played, then a card is chosen that has a value below the larger of the two.

Dropping a Card In case a suit cannot be obeyed, the general rule is that a card is dropped from a suit that, according to our statistical information, provides the largest improvement in the winning probability. The obvious situation is if suits becomes safe, otherwise the probability tables have to be consulted.

Playing Null Ouvert

In imperfect information games, the belief state space is the set of possible states the game can be in. In the worst case for $|S|$ as the number of states in S, the belief state space can grow exponentially in $|S|$. In Ouvert games, however, the belief state space (in view of the opponents) is much smaller. It is a game of almost full information, so that exhaustive search is more effective. To reduce the uncertainty for the opponents in the number of unknown hands down from $\binom{12}{10} = 66$ possible cards, we determine *good* cards that –under reasonable play– the declarer should not have been put.

Definition 4 (Good Card) *A card c in hand h is* good *if the dropping gain is smaller than the highest dropping gain of any card she has shown, i.e.,*

$$good(c) = 1 \text{ iff } drop(h, h \setminus \{c\}) < \max_{c' \in h} drop(h, h \setminus \{c'\}).$$

Once a card has being played, it is no longer unknown. For the remaining choices in the unknown cards, we call the glass box solver. If we have an unambiguous majority vote, we know which card to play. In case the vote is ambiguous, we take the rules from Null as a tie breaker. Besides evaluating all states in the belief space, we invest more efforts in the ordering of moves (reorder). The major observation we exploit is that in the tree search expert rules will consider more promising card proposals first.

```
decide_beliefs()
  nskat = nhand = 0;
  for (i=0;i<32;i++) vote[i] = 0;
  maxvote = 0;
  while (unknown)
    chosen1 = select(unknown);
    nskat |= (1 << chosen1);
    unknown &= ~(1 << chosen1);
    rest = unknown;
    while (rest)
      chosen2 = select(rest);
      nskat |= (1 << chosen2); rest &= ~(1 << card2);
      nhand = other_hand_or_skat & ~nskat;
      reorder();
      if (start == 1)
        if (!glassbox(hand0,hand,nhand))
          vote[proposed_card]++;
      if (start == 2) {
        if (!glassbox(hand0,nhand,hand))
          vote[proposed_card]++;
      new_skat &= ~(1 << chosen2);
    new_skat &= ~(1 << chosen1);
  for (i=0;i<32;i++)
    if (maxvote < vote[i]) maxvote = vote[i];
```

Figure 4: Selecting an opponents' card in Ouvert game.

We apply different orderings. For example, we avoid playing suits, in which the declarer is unbeatable, and prefer ones that enable the other opponent to play a card that beats her. We also favor cards that increase the degree of knowledge of the cards. While knowledge is computed for the belief space, turn transfers are computed for each of the state in the belief space. To enhance the computation together with good cards, we compute all possible transfers of play between the opponents.

We observed that opponent play is close to optimal. The informal argument we give is information-theoretic. With known safe and good cards the number of remaining cards being unknown to the opponent player is small, and, with general rules of play, the players can furtherly reduce. Based on the analysis of the transfer graph of opponent play and the options for dropping, safety analysis can go much deeper: for example a player's hand with three low cards and one A in a suit can only be beaten, if all the other cards are in one opponents hands; for 5 and 6 cards in one hand similar rules exist; in some cases a 2:2 partitioning in a suit is needed to win; and so on. At some stage within the game the skat is completely known, and from this moment onwards, the optimal game can be played. If one looks at the tricks needed to beat the player, then the clarifying tricks lead to a possible reordering/inclusion of tricks compared to the optimal play for the same hand.

In the implementation we maintain a bitvector of the 12 unknown cards for each player, and update the status of the cards on each trick, again using safe/good card analyses and possible turn transfers, as well as following the set of expert rules: 1) in case of a trivially flawed suit, transfer play immediately to opponent that can beat it; 2) transfer play to the opponent to put player in second position, even better if possible in the dropping suit; 3) if the above rules do not apply, change play to a non-flawed suit, that can overtaken by certain; 4) if none of the above rules apply, play the best card is the one that gets the declarer in second position.

Playing Trump Games

There are two main categories of Grands: a) *standard* grands (more than one J), in which strong Js and long suits secure a win, and b) *high-card grands*, in which the high-value cards (namely A and T) warrant sufficient points to satisfy the contract. Similarly, there is a distinction between *standard* Suit games (with more than 4 trumps), and b) high-card Suit games, in which even some high-valued trumps can be put.

The play of the declarer depends heavily on the category of the Grand. In standard grands she plays Js as soon possible to reduce the options for the opponents to gain control. In the case of a high-card grand, Js are played first, only if all 4 As are present. She will also play the suit first, which has the highest risk to be cut on by the opponents. Exceptions are if there are more than two ways to win.

After clarifying the status of the Js, the declarer usually tries to close gaps in his cards, so that virtual standing cards become standing. We tabled priorities 1–6 for this to happen for each suit, and propose the index of the first card in a suit to be played. In case of a tie the longest suit is chosen. Remaining ties are broken randomly.

We monitor and update the sum of points for all the players, and the cards being played in a data structure. In the beginning the value in the skat is known only to the declarer, but the opponents deduce information based the cards being played (especially, by looking at trick-openers and dropping cards). Exploiting the knowledge on bidding values and showing complement colors are strong weapons of the opponents. For each trick and each player, we evaluate if the game is won according to the contract. For the case that there are only standing cards left in the hand of the declarer, the game ends and all tricks assigned to her.

In standard Grands, the declarer first clarifies the status of the Js to avoid losing high-cards later on. There are exceptions, though. In case of 2 Js and 2-3 As at turn, usually a standing card without an A is played first. With 2 Js and 3 As, and a suit of at last two cards with A, this A is chosen. Other exception are two Js with 1 A and many standing cards in a color, or 3 Js, 1-2 As and more than 4 cards in one suit, etc.

In high-card Grands Js are used to take the turn. For each distribution of Js, we store the cards to be selected. The J in spades often is the best proposal as it hides the strength of the declarer. For the declarer we distinguish between selecting the first card and reacting to cards on the table. In case a suit cannot be obeyed, the general rule is that a card is dropped from that suit, which, according to our statistical information, provides the largest improvement in the winning probability. A high-value card is often kept, while lower cards are dropped. K and Q are frequently cashed, if the declarer is second in the trick. If a suit has to be obeyed, the smaller card is taken in second position.

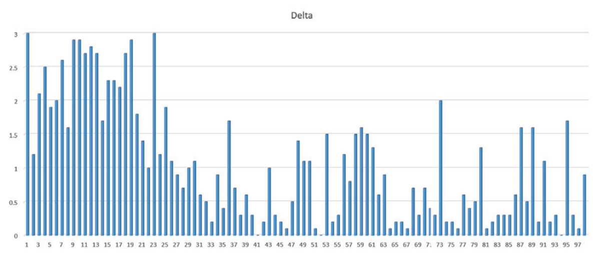

Figure 5: Prediction accuracy denoting the difference of winning in human play and computer play in the Nullspiel, partitioned along $P_w \in \{0\%, 1\% \ldots, \text{to } 100\%\}$.

There are similar rules for Suit game play (Gößl 2019), choosing a well-defined strategy to be followed.

Experiments

We implemented our Skat player in C/C++ using the GNU gcc compiler. For the experiments we used one core of an Intel(R) Core(TM) i7-7600U CPU @ 2.80GHz with 16GB RAM. The different versions of the game have a significant impact on the winning chances, which is reflected in the probability tables we apply (in form of plain arrays and STL hash maps).

We compared our performance with the Double-Dummy Skat Solver (DDSS) by Sebastian Kupferschmid (2003). DDSS is designed for trick playing only; it does not bid nor put. The performance of DDSS in analyzing open games stands out as a trademark. For the Nullspiel Kupferschmid measured a mean runtime of 20ms for analyzing a random Nullspiel game, which is remarkable. Our AI, however, improved this performance with an average runtime of about 5ms. Forced play of contracts in randomly deals, however, is an artificial assumption, so we decided to replay expert games. As DDSS has a different card encoding, we wrote a converter to provide human played games as input. The DDSS solver is fast, but the playing strength in the Nullspiel (with option partial observable play) was not convincing. Unfortunately, we also found bugs in the DDSS. We refactored the code and applied workarounds, but eventually we decided to compare with the Java player Fox1.2 (Furtak, Buro), which was more reliable. According to our knowledge, the solver code is based on DDSS anyway.

Our AI can play entire games, including bidding, game selection, skat putting and trick play. Computing the bidding value based on evaluating winning probabilities the $231 \cdot 66$ skats for each player and possible game type took about 0.03s per game. If the skat is not taken, the amount of uncertainty is higher and leads to the generation and usage of specialized probability tables for all the hand games, where the concept of *good* cards does not apply. Other than this, rule-based play and exploration algorithms remain the same. Figures 5 and 6 validate that the predicted winning probabilities are in good match with the human outcome in playging the Nullspiel and Grand. The evaluation of predicted winning probabilities in Suit games is similarly accurate.

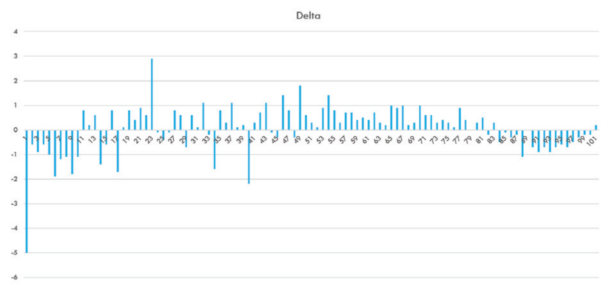

Figure 6: Prediction accuracy denoting the difference of winning in human and computer play in Grand, partitioned along $P_w \in \{0\%, 1\% \ldots, \text{to } 100\%\}$.

We next validated that the skats being put by the computer can be better than the ones of humans. Our evaluation criterion is running the glass box with both the human and the computer generated skats on 2.5M Null games (with a winning probability of 0.5 to 0.8). In less that 4h (221m and 229m, corresponding about 5ms per game), the total number of games won by the declarer in the glass box analysis wrt. human skat putting was 576K, and 623K for the computer skat, a significant lead for the AI.

Some experiences were made with other Skat engines. In the Nullspiel, with all respect, they all play rather poor and cannot match with human expert play, nor with our program's playing strength. Most of the systems were simply not designed for misere play.

Computer vs. Computer

In this experiment we took a wide selection of 60K expert Skat games of all different types and contracts. For Fox1.2 we converted the games into its so-called skat game format (sgf). Fox1.2 took about 19h (or 1.13s on average per game) to find the optimal game value together with the best cards to play. In contrast, our player took about 9.5h for the same task (0.52s on average), showing a speed-up of about factor 2. For both engines, deciding the game is much quicker: Fox1.2 required about 4.5h for the 60K games (250ms per game), while our AI took around 1h (or 63ms per game), showing a speed-up factor of about 4. We validated that the results were in match.

Computer vs. Human

For the qualitative analysis, we first chose some human Nullspiel games selected by an expert as being characteristic.

Putting Skat First, we were interested in the quality of putting the skat. For the hand ♢A;♢9; ♢7; ♡7; ♡K; ♡A;♠7; ♣8; ♣9; ♣K;— ♢8;♢J; ♡8; ♡9; ♡J; ♡Q;♠T; ♠J; ♠Q; ♣Q— ♢T;♢Q; ♢K; ♡T; ♠8; ♠9;♠K; ♠A; ♣T; ♣J; before taking the skat we obtain ♢: 72.8% win; ♡: 11.2% win; ♠: 100% win; ♣: 63% win. This results in a hand winning probability of $0.728 \cdot 0.112 \cdot 1 \cdot 0.63 = 5.1\%$ for the entire hand. After taking the skat, some winning probabilities for each possible skat putting option were: a. ♢A, ♡A : $1 \cdot 0.614 \cdot 1 \cdot 1 = 61.4\%$, b. ♢A, ♡K : $1 \cdot 0.523 \cdot 1 \cdot 1 = 52.3\%$, and c. ♡A ♡K : $0.728 \cdot 1 \cdot 1 \cdot 1 = 72.8\%$. (To determine the hand strength before skat taking, we average the winning probability over all possible skats.)

Null Games In the the game Null (23) the above set of given expert rules were implemented and first tested positively on a set of critical examples, selected by a world-class Skat player. Our play in the following deals matched the expert assessment. We distinguish between the player in first, second, and third position within a trick.

1. ♢J;♢9; ♢8; ♢7; ♣8; ♡K;♡Q; ♡8; ♡7; ♠8— ♢T;♢K; ♢Q; ♠A; ♠K; ♠J;♠9; ♣Q; ♣J; ♡9— ♢A;♣A; ♣K; ♣T; ♣9; ♣7;♠T; ♡A; ♡T; ♡J; Declarer puts ♡K and ♡Q; In proper opponent play he still loses. 1st Trick: ♣8; ♣Q; ♣A; 2nd Trick: ♠T; ♠8; ♠A; 3rd Trick: ♠9; ♠7, ♢A; 4th Trick: ♠J; ♠Q;
2. ♣T;♠Q; ♠J; ♠9; ♡T; ♡K;♡Q; ♡J; ♢T; ♢K— ♡A;♡9; ♡8; ♡7; ♢Q; ♢9;♢7; ♠T; ♠7; ♣8— ♢8;♢J; ♢A; ♠8; ♠K; ♠A;♣K; ♣Q; ♣9; ♣7; Skat already put. Declarer loses. 1st Trick: ♣T; ♣8; ♣ K; 2nd Trick: ♠7; ♢K; ♢ Q; 3rd Trick: ♠9; ♢8; ♢*T*, ♢9 4th Trick: ♡T; ♡9; ♠A 5th Trick: ♡J; ♡8; ♠K 6th Trick: ♡Q; ♡7; ♢A 7th Trick: ♡K; ♡A;
3. ♣K;♣Q; ♠J; ♠8; ♡A; ♡T;♡Q; ♢T; ♢Q; ♢J— ♣A;♣J; ♣T; ♣9; ♣7; ♠Q;♠9; ♠7; ♡8; ♡7— ♣8;♠A; ♠T; ♠K; ♡J; ♡9;♢K; ♢9; ♢8; ♢7; Skat already taken, declarer in MH loses, first cards 1st Trick: ♠8; 2nd Trick: ♣8; and 3rd Trick: ♠J (better than ♣Q); rest simple.

Queen with Three For Ouvert play, therefore, one set of games we considered, consisted of the declarer having a weakness of 7,8,Q (or 7,9,Q), which may be exploited. When we selected games, in which the human player won, we found several games in which the computer opponents turn the game. Our analysis of the remaining data set of 65 534 lost games including a possible line of play required 23m19s (about 21ms each). The top level glass box search took 5ms on average for analyzing a single game. We identified 3 902 games that were saved.

Declarer-First Games Next, we analyzed games, in which the declarer issues the first card. We found over 17K expert games in the database, in which she lost. By refined skat putting, we experienced that about 4K games could be saved, even with optimal game play of both the player and the opponents. The time for analyzing all games (including a certificate of play in case the game was won) was 4m21s. Using a non-optimal reflex player, the value dropped to 3 901 games and 3m30. Using human skat putting, the number of saved games went down to 960, and the time for analyzing 2min8s. This corresponds to about 7ms per game.

Null Ouvert Games Of the total of 290 084 critical NO-Games with a predicted winning probability of $> 50\%$, the human opponents won $217\,193 = 74.9\%$ games while, our computer opponents won $243\,365 = 83.9\%$ games. This is a surplus of 26 172 games. For a fair comparison in both cases the skat was fixed to what the human declarer had put.

Anlyzing about 300T games, including the generation of a line of trick play, took 4231m, which means that each game is played in less than 1s. This indicates that the opponent players significantly outperforms human play.

Table 1: Results in NO Games.

GlassBox	WonHuman	WonAI	Number	%
0	0	0	186 656	64.3%
0	0	1	30 537	10.5%
0	1	0	56 709	19.5%
0	1	1	16 182	5.6%

Table 2: Result in Null Games.

GlassBox	WonHuman	WonAI	Number	%
0	0	0	382 123	19.7%
0	0	1	495 909	25.5%
0	1	0	222 597	11.5%
0	1	1	843 226	43.4%

From the 2.5M Null games with a winning probability between .5 and .8 the optimal open card game solver lost 1 943 855 games, which is 77.1%. Of these, the human saved 45.2% games, the PC 31.1%. Sources of inferior play have been removed, so that the AI saved many more games, but still does not yet exceed human performance. We identified the weakness of the player in the rules for color change, which have to be carefully coded, since they need to be updated dynamically. For example, if an opponent issued card is obeyed by the declarer, but one does not have a small card that takes her down, a color change is needed. Small cards include 7 or 8, 8 and 9, 8 and T, 9 and T, 9/T/B (which beats 7, 8/9, Q), or 9/T/Q (beats 7, 8/9, K).

Conclusion

In this paper we presented a novel Skat AI that incorporates distilled human game information, expert rules, aggregates winning probabilities for bidding, game selection, as well as skat putting, and which exploits fast tree exploration. According to the glass box evaluation at the current stage of implementation, skat putting is better than humans in the Nullspiel, equal in Grand games, and slightly worse in Suit games. We expect to put the skat better than the human in all types of games soon. More importantly, the AI plays Null Ouvert significantly better than human experts, and the exploration is significantly faster than in other Skat playing tools. By refined expert rules of play, while updating knowledge of the players, we are optimistic that the general strength of our AI Skat player will eventually exceed human tournament play. While our program is primarily designed to play Skat, some algorithmic aspects can transfer to other imperfect information card games, including Tarots, Bridge and Hearts. We are also looking forward to see whether or not our use of a large body of statistical expert information carries over to other incomplete information games.

Acknowledgement This research work would not have been possible without the ingenious help of Rainer Gößl, a Skat expert and world-class caliber player.

References

Allis, L. V.; van der Meulen, M.; and van den Herik, H. J. 1994. Proof-number search. *Artificial Intelligence* 66:91–124.

Allis, L. V. 1998. A knowledge-based approach to connect-four. the game is solved: White wins. Master's thesis, Vrije Univeriteit, The Netherlands.

Bowling, M.; Burch, N.; Johanson, M.; and Tammelin, O. 2017. Heads-up limit hold'em poker is solved. *Commun. ACM* 60(11):81–88.

Campbell, M.; A. J. Hoane, J.; and Hsu, F. 2002. Deep blue. *Artificial Intelligence* 134(1-2):57–83.

Furtak, T. M. 2013. *Symmetries and Search in Trick-Taking Card Games*. Ph.D. Dissertation, University of Alberta.

Gasser, R. 1995. *Harnessing Computational Resources for Efficient Exhaustive Search*. Ph.D. Dissertation, ETH Zürich.

Ginsberg, M. 1999. Step toward an expert-level bridge-playing program. In *IJCAI*, 584–589.

Gößl, R. 2019. *Der Skatfuchs – Gewinnen im Skatspiel mit Mathematische Methoden*. Selfpublisher. Dämmig, Chemnitz, Available from the Author or via DSKV Altenburg.

Grandmontagne, S. 2005. *Meisterhaft Skat spielen*. Selfpublisher, Krüger Druck+Verlag.

Harmel, S. 2016. *Skat–Zahlen*. Klabautermann-Verlag, Pünderich (Mosel).

Keller, T., and Kupferschmid, S. 2008. Automatic bidding for the game of skat. In *KI*, 95–102.

Kinback, T. 2007. *Skat-Rätsel – 50 lehrreiche Skataufgaben mit Lösungen und Analysen*. Books on Demand, Norderstedt.

Kupferschmid, S., and Helmert, M. 2006. A Skat player based on monte-carlo simulation. In *Computers and Games*, 135–147.

Kupferschmid, S. 2003. Entwicklung eines Double-Dummy Skat Solvers mit einer Anwendung für verdeckte Skatspiele. Master's thesis, Albert-Ludwigs-Universität Freiburg.

Lasker, E. 1929. *Das verständige Kartenspiel*. August Scherl Verlag, Berlin.

Lasker, E. 1938. *Strategie der Spiele – Skat*. August Scherl Verlag, Berlin.

Long, J. R. 2011. *Search, Inference and Opponent Modelling in an Expert-Caliber Skat Player*. Ph.D. Dissertation, University of Alberta.

Quambusch, M. 1990. *Gläserne Karten – Gewinnen beim Skat*. Stomi Verlag, Schwerte Rau Verlag, Düsseldorf.

Schaeffer, J.; Björnsson, Y.; Burch, N.; Kishimoto, A.; Müller, M.; Lake, R.; Lu, P.; and Sutphen, S. 2005. Solving checkers. In *International Joint Conference on Artificial Intelligence*, 292–297.

Schettler, F., and Kirschbach, G. 1988. *Das große Skatvergnügen*. Urania Verlag, Leipzig, Jena, Berlin.

Schubert, H. 1887. *Das Skatspiel im Lichte der Wahrscheinlichkeitsrechnung*. J. F. Richter, Hamburg.

Silver, D., and et al., A. H. 2016. Mastering the game of go with deep neural networks and tree search. *Nature* 529:484.

Silver, D.; Hubert, T.; Schrittwieser, J.; Antonoglou, I.; Lai, M.; Guez, A.; Lanctot, M.; Sifre, L.; Kumaran, D.; Graepel, T.; Lillicrap, T.; Simonyan, K.; and Hassabis, D. 2017. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. Technical Report 1712.018, arxiv.

Wergin, J. P. 1975. *Wergin on Skat and Sheepshead*. Wergin Distributing, Mc. Farland, USA.