Challenging Human Supremacy in Skat – Guided and Complete And-Or Belief-Space Tree Search for Solving the Nullspiel

Stefan Edelkamp

Department of Informatics
King's College London

Abstract. After impressive successes in deterministic and fully-observable board games to significantly outclass humans, game playing research shifts towards nondeterministic and imperfect information games, where humans are persistently better. In this paper we devise a player that indicates computer supremacy for the *Nullspiel*, a misere variant of Skat, in which the declarer has to lose all tricks. We combine knowldege elicitation in several million games, together with expert rules, and engineered tree exploration that eventually leads to the *Guided and Complete And-Or Belief-Space Search*. We provide a complete player, with effective solutions for bidding, skat putting and for playing the different variants of the game.

1 Introduction

Many known fully-observable deterministic combinatorial games are either solved [6, 12, 1] or computers significantly outperform human play [4]. 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 [13]. 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 [14].

In contrast, card games with randomness (in the deal) and imperfect information (due to hidden cards) are still considered a major challenge to AI game playing technology. One recent exception is Heads-Up Limit (Texas) Hold'em Poker, where computational game theory and extensive search have established an ϵ-optimal Nash equilibrium [3].

For trick-based card games, the situation is unsatisfactory. After visible early advances in playing Bridge [7], research progress has slowed down considerably, even though computer programs (like Wbridge and Jack) are under continuous development. In Skat, an international three-player card game using a deck of 32 cards, with a deal of 10 cards for each player and two left-overs forming the so-called *skat*, there is slightly more recent research being published. Work on an efficient open card Skat player [9] went into an expert-level Skat player [11]. Symmetries and refined search algorithms have been studied by [5]. For world-class play of Skat, however, there is still a significant gap. We are developing a program that – in the long run – is expected to outperform world-class human play. As a first result on this research avenue, we present a program

that is specifically designed to play the *Nullspiel*. This variant is particularly interesting, as existing Skat programs tend to play rather weak. The Nullspiel is a variant of the Skat game, in which in order to win the contract, the declarer has to lose all ten tricks. There are game variants with open cards, and with not taking the skat.

In this paper we contribute a new approach to solving the Nullspiel. We compile the information of over six million games in a clever way, and use a combination of expert rules, aggregrated statistical information on winning probabilities and information gain, as well as high-performance exploration algorithms. For Null Ouvert the Guided and Complete And-Or Belief-Space Search algorithm shows that computer play outperforms human play.

The paper is structured as follows. First, we explain the rules of the Skat game in general and Nullspiel in particular. Next, we shed light on the bidding and Skat putting process, exploiting the expert wisdom that has been acquired over the years in the form of a database with millions of played games. We walk through the versions of the Nullspiel, which lead to different player proposals. For Null, we mainly rely on statistical information and expert rules, while for Null Ouvert we propose an engineered and sound guided but complete and-or tree search algorithm, where the exploration has been been enriched with preferred knowledge moves (e.g., to reduce the amount of uncertainty in the remaining cards). We give experimental results to show that our program is superior to existing technology, and that it plays better than humans.

2 Skat and the Nullspiel

The rules of Skat go back to Hempel around 1848. Competitive Skat is defined by the International Skat Player Association[1], The game is played with three players. A full deck has 8 cards (A, 10, K, Q, J, 9, 8, and 7) in all four suits (♣, ♠, ♡, ♢). After shuffling, each player receives 10 cards, while the skat consists of the two left-overs. There are three phases of the Skat game: the bidding stage, taking and putting the skat, and the actual play for tricks. The *declarer*, which went first in the *bidding*, is playing for a win against the remaining two *opponents*. He is allowed to strenthen his hand by taking the two left-overs and putting two other ones (could be the same ones), which are counted to his favor.

The games being played and bid for are *Trump*, which includes *Grand* and *Suit* (♣, ♠, ♡, or ♢), as well as the *Nullspiel*, a misere, trick-avoiding variant of the game. A related card games is *Hearts*, where players avoid winning certain penalty cards in tricks, and winning tricks altogether.

Moreover, there are four variants of the Nullspiel: *Null* (bidding value 23), *Null Hand* (35), *Null Ouvert* (46), and *Null Ouvert Hand* (59), where *Ouvert* forces the declarer to show all his cards prior to the play, and *Hand* forbids the declarer to take the skat. Different to playing Trump, the declarer that wins the bidding, must lose all tricks. In this variant according to the expert judgement[2] most computer card game progams play badly.

[1] available at `www.ispaworld.info`

[2] See `www.skatfuchs.eu` for a list of studied programs

```
safe_cards(hand, played)
  safe = 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++;
        safe |= card;
      else
        if (counter == 0) break;
        counter--;
  return safe;
```

Fig. 1. Computing safe cards in the hand of the declarer with respect to a set of already played cards.

In the Nullspiel the ordering of cards is A, K, Q, J, 10, 9, 8, and 7. If the declarer gets any trick, he loses. To the contrary, he wins by certain if his hand is *safe*.

Definition 1 (Safe Card). *A declarer card is* safe*, if all gaps smaller than it are supported by at least the matching number of cards below them, with a special case for the declarer's turn in the first trick, where one extra play card is requested. The declarer's hand is* safe *if all cards are safe.*

The definition of a safe hand includes the cards in the skat, and can accommodate the cards being played. A concise implementation of the decision procedure to compute safe cards (for all but the declarer's first card) is provided in Fig.1 (the code assumes bitvector sets introduced later on, and refers to the pseudo-code in Kupferschmid's masters thesis).

Different strategies for the declarer and his opponents depend on the position of the players within the trick and dominate expert play. Implicit rules like *Shortest Suit – Smallest Card First* indicate the fundamental importance of colaboration between the two opponents for maximizing the knowledge exchange. Such hidden rules are seeming 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 of the opponents is preferred, if it is a safer way to win the game. 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.

In each suit, we have eight cards and $2^8 = 256$ different selections of cards that form a pattern (see Fig. 2). As the binary representation of the position in the vector and the 0/1 pattern are identical, given the bitvector for a given hand, we can extract the probability value in time $O(1)$ by bit masking, bit shifting and bit reversal operations on the vector. For each of these pattern we determine the probability of winning, using a multinomial distribution refined with the winning probabilities in the expert games. The probabilities are stored in tables, and the winning probabilities P_w among all the suits are multiplied as an estimate for the overall winning probability, i.e.,

```
  0. ........
  1. .......7
  2. ......8.
  3. ......87
  4. .....9..
  5. .....9.7
  6. .....98.
  7. .....987
    [...]
255. AKQJT987
```

Fig. 2. Suit patterns to address probabilities and proposed cards; the index in binary matches the bitvector to its right.

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

3 Bidding and Putting Skat

For bidding and putting the skat we, therefore, follow the suggestion of Emanual Lasker [10] (primarily known as the second and only German world champion in Chess), and combine statistics on winning probabilities in each suit.

Bidding In case of no *jump bids*, the bidding stage follows a predefined order of calling numbers corresponding to the value of the game $(18, 20, 22, 23, 24, 27, 30, 33, 35, etc.)$. The bidding value of Suit and Grand games depends on the distribution of Jacks, for the Nullspiel variants it is fixed constant (see above).

There is some information on the card distribution that can be extracted by denoting on when the bidding has stopped, as this indicates, e.g., in which suits the players are strong. Different bidding strategies have been proposed; in the literature, among others we find neural networks [9], nearest neighbor search [8], and statistical analyses of human games and single-agent game tree search for bidding, discarding, and game type announcements [11].

In contrast to these classical learning and game-theoretical approaches, for each of the possible deals, we exploit rather flat statistical knowledge on the individual strength of the cards in hand that we have extracted from millions of expert games. The probabilities to win a chosen game is estimated and multiplied with the expected value of the game, which for the bidding is maximized[3].

Given that for each type of game being played there are $3\binom{32}{10,10,10,2} = 8\,259\,883\,225\,513\,920$ different deals (choice of the cards and including selecting the turn) a lookup table, however, is seemingly too large and, more importantly, even for many millions of stored games way too sparse to retrieve values for each game.

Thus, for the Nullspiel for bidding we use the winning probabilities multiplied to the value of the game (for the other types of games we use a refined hash function to store and address similar entries. Such hash functions select a set of features to generate

[3] For extracting this information we refer to the data stored in *Skatfuchs* that offers distilled winning probabilities to the public, see `www.skatfuchs.eu` and the according online forum `www.32karten.de`.

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

Skat Putting Once the skat is 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 2 (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. This also applies to length four suits with A and K: an example is ♣:7, 9, K, A and ♡: 7, K, where it is better to put ♣ K and A into the skat instead of the K in ♡.

As a refinement for 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.

We validated that for the Nullspiel the deviation of the winning probabilities in this bidding and skat putting strategies compared to the expert play in the large set of six million games was at most 3% (see Figure 5)[4] For us this observation came as a surprise as the crude approximation of winning chances by multiplying factored 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.

4 Null

The algorithm for hidden card games like Null and Null Hand we propose 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 and 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 lookup table. The algorithm consisting of a set of expert rules roughly looks as follows.

For the first card choice of the declarer we also use a table addressed with patterns. The first table gives the probability of winning by choosing this card, the second one

[4] We do have mathematical evidence that the accuracy of the prediction is bounded at most 4%.

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.

4.1 Opponents' Choice

First for the opponents.

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, then play it immediately. If there are two or more short colors, choose the one with the higher winning probability. For example in (♣K, ♣Q, ♠J, and ♠8) choose 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 ♢K being played and (♢A, ♢Q, ♢7) in hand, play ♢A and then play ♢7, not ♢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 he may have put some into the skat.

Change of Suit If one does not have any suit with small cards, then 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 10, or 9 and 10, or 9 and 10 and J, or 9 and 10 and Q); c) the other opponent can't drop all cards on the played suit; d) one has a singleton, which one can drop.

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 a 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.

4.2 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 there is 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.

5 Null Ouvert

Null Ouvert is a game of almost full information, at least for the opponents, so that exhaustive exploration algorithms can be effective. For the analysis, we implemented an engineered And-Or tree solver to determine the game-theoretical value of the game. Figure 3 shows the implementation that includes transposition table pruning, the detection of equivalent cards, and the analysis of safe cards. The game is embedded into a loop of interactive play, in which we can exchange human and computer input, optimal and suboptimal strategies.[5]

To represent hands, for the skat and the played cards we use bitvectors (in form of unsigned integers). This allow Boolean set operations: `&` , $\sim$, 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 (short `first(x)`), and the last card as `TZCOUNT(x)` (short `last(x)`). For the representation of a *state* of the game, we chose the union of the three hands.

The backtracking algorithm in Fig. 3 is not too difficult to understand. The code fragments for the opponents (Or nodes OR1 and OR2) are conceptually similar, with the outcomes 0 and 1 reversed. It returns the game-theoretical value (lost, won) of the game at a given node in the And-Or search tree, with an And-node (AND) referring to the player and an Or-node referring to one of the two opponents (OR1 and OR2).

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 And-Or search. The variables that are set to some value are set back to their original 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 Boolean reasoning to generate an optimal strategy. Pruning takes place as part of the Boolean formulas as only parts of it (the principal variation) need to be evaluated.

[5] We also implemented *proof-number search* [2], but given the involved handling of transpositions and the experienced higher efforts per node, in our implementation it was less efficient and reliable than the And-Or solver.

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

Fig. 3. And-Or Tree Search with transposition table pruning for the declarer's turn (code for the opponents' turn is similar).

To avoid a common misunderstanding: an AND node is false in Boolean terms (which means that the player wins) if one of it successors is false (1), otherwise it is true (0), which means that the player loses. In case all cards are on the table, the trick is evaluated and the game is either stopped or continued with the player that according to the rules is the winner of the trick. To avoid problems with the ongoing trick, we use the transposition table only after a trick has been played. This is sufficient to encode the entire state into 32 bits, using the skat that do not change over time to denote the turn.

To maintain the order of play within the trick, which is important to evaluate its outcome, we use explicit indices ($i[0..2]$) of the cards being played instead of a bitvector.

6 Guided and Complete And-Or Belief-Space Search

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 state in S, the belief state space can be as large as $2^{|S|}$. In open card games like NO, however, the belief state space (in view of the opponents) is much smaller.

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 circumstances– the declarer cannot have been put. Of

Definition 3 (Good Card). *A card c in hand h is* good *if the dropping gain is smaller than the highest dropping gain of any card he has shown, i.e.,*

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

```
decide_beliefs()
  new_skat = new_hand = 0;
  for (i=0;i<32;i++) vote[i] = 0;
  maxvote = 0;
  while (unknown) {
    card1 = last(unknown);
    new_skat |= (1 << chosen_card1);
    unknown &= ~(1 << chosen_card1);
    rest = unknown;
    while (rest)
      card2 = last(rest);
      new_skat |= (1 << card2);
      rest &= ~(1 << card2);
      new_hand = otherhandorskat & ~(new_skat);
      reorder();
      if (start == 1)
        if (!AOS(hand0,hand,new_hand) && proposed)
          vote[proposed_card]++;
      if (start == 2) {
        if (!AOS(hand0,new_hand,hand) && proposed)
          vote[proposed_card]++;
      new_skat &= ~(1 << chosen_card2);
    new_skat &= ~(1 << chosen_card1);
  }
  for(int i=0;i<32;i++)
    if (maxvote < vote[i])
      maxvote = vote[i];
```

Fig. 4. Guided and Complete And-Or Belief-Space Tree Search for selecting an Oppents' Card in Null Ouvert.

Of course, once a card is being played, it is no longer unknown. For the remaining choices in the unknown cards, we call the solver.

If we have an unambiguous majority vote, the player knows, which card to play. In case the vote is ambigous, we take the rules from Null as a default. Besides evaluating all states in the belief space, we take 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.

We use different ordering rules. For example, we avoid playing suits in which the declarer is unbeatable, and we prefer cards that enables the other opponent to play a card that beats the declarer. We also favor playing cards that increase the degree of knowlege of the cards. While knowledge is computed for the belief space, transfers are computerd for each of the state in the belief space. To enhance the computeation together with good cards, we compute all possible transfers of play between the opponents, based on their cards beforehand.

We observed that the opponent player playing with uncertainty (of skat putting) is almost good as the optimal player. The informal argument we give is information-theoretic. With known safe and good cards the number of remaining cards being uncertain to the opponent player is small enough that general rules of play and some tricks can clarify. Based on the analysis of 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(ce) 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 there on, the optimal game is 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 quo of the cards on each trick, again using safe/good card analyses and possible 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 MH, even better if possible in the dropping suit; 3) if the above rules do not apply, transfer play to opponent in a non-flawed suit, that he can overtake by certain; 4) if none of the above rules apply, play the best card to get the player in MH.

7 Hand Games

The degree of uncertainty for all players in a *hand* game (in which the skat is not taken) is significantly higher, as there is no control of the player on putting the skat. On the other hand, the skat is uniformly sampled, so that no bias to it needs to be assumed. For Null Hand and Null Ouvert Hand the number of tricks is less not sufficient to extract full knowledge on the cards being put as the assumptions on *good* cards does not apply. Other than this, rule-based play and Guided and Complete And-Or Belief Space Search remains the same.

8 Experiments

We implemented our Skat Nullspiel player in C using the GNU gcc compiler (optimized with option -O3), Transposition table pruning is chosen as a default. The different versions of the game have a significant impact on The potential winning chances, which are reflected in the probability tables that we apply. For the experiments we used one core of an Intel(R) Core(TM) i7-7600U CPU @ 2.80GHz with 16GB RAM. We are working on a server for a head-to-head comparison, but for the time being the input are large databases of human expert games (maintained in the courtesy of our domain expert).

9 Bidding

Figure 5 validates that the prediction of the winning probabilities for a hand via the multiplication of the winning probabilities in all suits matches human play quite well.

9.1 Skat Putting

We first validated that the skats being put by the computer are better than the ones put by the human's. Our evaluation critieria is running the glass box (our term for double-dummy solver) with both the human and the computer generated skats on 2.5 million 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 576000, while for the computer skat 623154, which is an indication of a significant improvement.

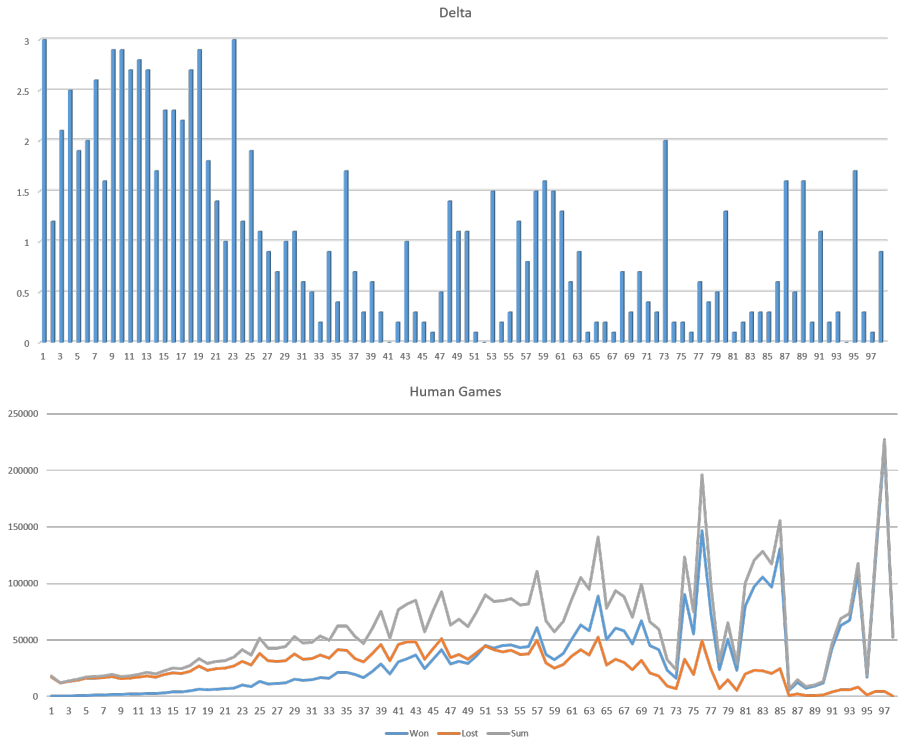

Fig. 5. (top) Prediction accuracy denoting the difference of winning in human play and in multiplying suit probabilities in the Nullspiel, partitioned along the predicted winning probabilities of $0\%, 1\% \ldots,$ to 100%. (bottom) Real games being played, where the set of over six million games is also partitioned along the predicted winning probabilities of $0\%, 1\% \ldots,$ to 100%.

9.2 Computer vs. Computer

We compared our player with the Double-Dummy Skat Solver (DDSS) implemented in C++ by Helmert and Kupferschmid. Thanks to the authors, the systems have been made available in source code. Unfortunately, we found a severe bug, manifesting in games in which it is not the declarer's not turn, so that we had to shift the players cyclically. For open games we refactored the code, to reduce the code (without the static tables) from about 8000 to about 1500 LOC.

DDSS is designed only for the trick playing stage, it does not bid or put. It also has a different card encoding and input format, so that we wrote a converter of our database input to provide human played games as input.

The playing strength of DDSS in the Nullspiel (option partial observable play), was pretty bad (it is better in Trump games), so that we decided not to dwell on its quality[6].

[6] We know Sebastian Kupferschmid well, so that we do not want to affront him for his otherwise great though academic implementation work. If requested by the reviewers, we can add data in the final version of the paper

Similar experiences were made by our domain expert with other Skat engines[7]. 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 are simply not designed well for misere play.

The performance of DDSS in analyzing open games to determine their game-theoretical value however, still stands out as a trademark. For the Nullspiel (in his masters' thesis) Kupferschmid states an average runtime of 0.02s for analyzing a Nullspiel, which is a remarkable achievement. Still, our algorithm improves this performance with an average runtime of about 5ms (4-fold improvement). The difference in computer technology does not make up for the difference, as the speed on one core has not increased much.

9.3 Computer vs. Human

For the qualitative analysis, we chose different sets of games played by humans. By the limited choice of Nullspiel games we neglected bidding[8], and simply played the game that the humans chose.

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;♠10; ♠J; ♠Q; ♣Q— ♢10;♢Q; ♢K; ♡10; ♠8; ♠9;♠K; ♠A; ♣10; ♣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*0.112*1*0.63 = 5.1% for the entire hand. After taking the skat, the winning probabilities for each possible skat putting option: 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 In the the game Null (23) the above set of given expert rules were implemented and first tested positively on a larger set of selected examples, which were judged critical by a world-class Skat player. Our play in the following deals matched the expert assessment. We distinguish between the player in first (VH), second (MH), and third position (HH) within a trick.

1. ♢J;♢9; ♢8; ♢7; ♣8; ♡K;♡Q; ♡8; ♡7; ♠8— ♢10;♢K; ♢Q; ♠A; ♠K; ♠J;♠9; ♣Q; ♣J; ♡9— ♢A;♣A; ♣K; ♣10; ♣9; ♣7;♠10; ♡A; ♡10; ♡J; Player in VH puts ♡K and ♡Q; In proper opponent play he still loses. 1st Trick: ♣8; ♣Q; ♣A; 2nd Trick: ♠10; ♠8; ♠A; 3rd Trick: ♠9; ♠7, ♢A; 4th Trick: ♠J; ♠Q;
2. ♣10;♠Q; ♠J; ♠9; ♡10; ♡K;♡Q; ♡J; ♢10; ♢K— ♡A;♡9; ♡8; ♡7; ♢Q; ♢9;♢7; ♠10; ♠7; ♣8— ♢8;♢J; ♢A; ♠8; ♠K; ♠A;♣K; ♣Q; ♣9; ♣7; Skat already put. Player in VH loses. 1st Trick: ♣10; ♣8; ♣ K; 2nd Trick: ♠7; ♢K; ♢ Q; 3rd Trick: ♠9; ♢8; ♢10, ♢9 4th Trick: ♡10; ♡9; ♠A 5th Trick: ♡J; ♡8; ♠K 6th Trick: ♡Q; ♡7; ♢A 7th Trick: ♡K; ♡A;

[7] There are many computer programs assessed at *skatfuchs.eu*

[8] The eager reader is referred to *skatfuchs.eu* for an weighted prediction of winning probabilities for all different bidding options.

3. ♣K;♣Q; ♠J; ♠8; ♡A; ♡10;♡Q; ♢10; ♢Q; ♢J— ♣A;♣J; ♣10; ♣9; ♣7; ♠Q;♠9; ♠7; ♡8; ♡7— ♣8;♠A; ♠10; ♠K; ♡J; ♡9;♢K; ♢9; ♢8; ♢7; Skat already taken, player in MH loses, first cards 1st Trick: ♠8; 2nd Trick: ♣8; and 3rd Trick: ♠J (better than ♣Q); rest simple.

Random Games To compare speed with DDSS, we randomly sampled deals and solved them: together with skat taking and putting for 10 000 Null Ouvert games we measured a run time of 59.8s; 2 773 games were won. When we looked at Null Ouvert Hand, the runtime reduced to 38.3s and only 113 being won. By not taking the skat, the strength of the player's card is weakened, so that the And-Or tree gets pruned more. We conclude that forced play in random deals is an artificial assumption, so in the following we prefer to look at critical games.

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 sometimes can 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 game including generating the certificates for one line of play required 23m19s (about 21ms each). The top level glass box search took 5ms for analyzing a single game. We identified 3 902 games that were saved.

Player-First Games Next, we analyzed games, in which the player that wins the bidding is first in selecting a card (in VH). According to this criteria we found over 17 000 expert games in the database, in which the first player also lost. By refined skat putting, we experienced that about four thousand games could be saved, even with optimal game play of both the player and the opponents. The time for analysis all games (including a certificate of play in case the game was won) was 4m21s. Using a non-optimal but reflex player, the value dropped to 3 901 games and 3m30. Using the skat putting strategy of the human players, the number of saved games went down to 960, and the time for analyzing 2min8s. This corresponds to about 7ms per game.

9.4 Null Ouvert Games

Of the total of 290 084 critical NO-Games with a predicted winning probability of $>$ 50% with the same skat put, the human opponent players won 217 193 $=$ 74.9% games while the opponent in our Guided and Complete And-Or Belief Space Search engine won 243 365 $=$ 83.9% games. This is a plus of 26 172 games. For a fair comparison in both cases we fixed the skat to what the human declarer had put.

The analyses of the about 300 thousand games, including the generation of a line of play took 4231m, which means that each game is played in less than a second. The result exceeded the expectations as the opponent players significantly outperforms human play.

Table 1. Null-Ouvert Games in Guided and Complete And-Or Belief Space Search that the declarer lost in glass box play.

WonAOS	WonHuman	WonPC	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. Null Games that the glass box lost.

WonAOS	WonHuman	WonPC	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%

9.5 Null Games

From the 2.5 million Null games with a winning probability between .5 and .8 the glass box lost 1 943 855 games, which is 77.1%. Of these, the human saved 45.2% games, the PC 31.1%.

Some sources of inferior play have been removed, so that the computer saves many more games, but the computer player still does not yet exceed human performance. We identified the weakness of the player mainly to resist in the rules for color change, which has to be carefully coded, since the rules have 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 fails him/her, a color change is needed. Small cards include a 7 or an 8, an 8 and a 9, an 8 and a 10, a 9 and a 10, a set 9/10/B (which beats 7, 8/9, D), or a set 9/10/D (beats 7, 8/9, K).

10 Conclusion

We presented a AI player for Skat that incorporates distilled expert rules, aggregates winning probabilities, and exploits a fast tree exploration. Skat putting was superior to, and the opponents dealing with partial information play the Nullspiel significantly better than human experts. The and-or search was significantly faster than other refined search variants. The top-level glass-box search determining the value of the game required about 5ms per game on the average.

Up to bidding and putting skat, the declarer is mainly reactive: For dropping cards s/he maximizes the gain in winning probabilities. By updating the knowledge of the opponents, good cards and trick transfer, actual play can be organized almost optimally, in the sense that a play that is equivalent to the optimal one is inmplied.

While our program is primarily designed to play Skat, some algorithmic aspects might transfer to other imperfect information (card) games, including bridge and hearts. We are also looking forward to see whether some ideas like using expert information both the play and in move order carry over to other incomplete information games, like Starcraft.

References

1. L. V. Allis. A knowledge-based approach to connect-four. the game is solved: White wins. Master's thesis, Vrije Univeriteit, The Netherlands, 1998.
2. L. V. Allis, M. van der Meulen, and H. J. van den Herik. Proof-number search. *Artificial Intelligence*, 66:91–124, 1994.
3. Michael Bowling, Neil Burch, Michael Johanson, and Oskari Tammelin. Heads-up limit hold'em poker is solved. *Commun. ACM*, 60(11):81–88, 2017.
4. M. Campbell, Jr. A. J. Hoane, and F. Hsu. Deep blue. *Artificial Intelligence*, 134(1-2):57–83, 2002.
5. Timothy Michael Furtak. *Symmetries and Search in Trick-Taking Card Games*. PhD thesis, University of Alberta, 2013.
6. R. Gasser. *Harnessing Computational Resources for Efficient Exhaustive Search*. PhD thesis, ETH Zürich, 1995.
7. M. Ginsberg. Step toward an expert-level bridge-playing program. In *IJCAI*, pages 584–589, 1999.
8. Thomas Keller and Sebastian Kupferschmid. Automatic bidding for the game of skat player. In *KI*, pages 95–102, 2008.
9. Sebastian Kupferschmid and Malte Helmert. A skat player based on monte-carlo simulation. In *Computers and Games*, pages 135–147, 2006.
10. Emanuel Lasker. *Strategie der Spiele Skat*. Scherl, Berlin, 1938.
11. Jeffrey Richard Long. *Search, Inference and Opponent Modelling in an Expert-Caliber Skat Player*. PhD thesis, University of Alberta, 2011.
12. Jonathan Schaeffer, Yngvi Björnsson, N. Burch, Akihiro Kishimoto, Martin Müller, Robert Lake, Paul Lu, and Steve Sutphen. Solving checkers. In *International Joint Conference on Artificial Intelligence*, pages 292–297, 2005.
13. David Silver and Aja Huang et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529:484, 2016.
14. David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, and Demis Hassabis. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. Technical Report 1712.018, arxiv, 2017.

11 Appendix

Further examples for critical Nullspiel games, including a so-called *hidden drop*.

1. ♣7;♣9; ♠K; ♠A; ♡10; ♡J;♡Q; ♡K; ♢7; ♢A— ♣8;♠7; ♠8; ♠9; ♠J; ♡7;♡8; ♡9; ♢8; ♢9— ♣10;♣J; ♣K; ♠10; ♠Q; ♡A;♢10; ♢J; ♢Q; ♢K;
2. ♠7;♡Q; ♣J; ♠A; ♣A; ♠10;♠K; ♣Q; ♠9; ♡9— ♢J;♡A; ♢A; ♣7; ♠J; ♢Q;♠Q; ♣9; ♡8; ♢K— ♢9;♡K; ♢8; ♠8; ♣8; ♡10;♡J; ♢7; ♡7; ♣10;
3. ♢A;♢10; ♢J; ♣10; ♣K; ♠10;♠Q; ♡10; ♡K; ♡9— ♣Q;♣9; ♣7; ♢K; ♢Q; ♢9;♢7; ♠J; ♠9; ♠7— ♣J;♣8; ♢8; ♠8; ♠K; ♡7;♡8; ♠A; ♣A; ♡A;
4. ♣J;♠10; ♡10; ♠7; ♢K; ♣10;♢J; ♣A; ♡9; ♢9— ♠A;♠9; ♣K; ♠Q; ♢10; ♢Q;♢8; ♢A; ♠8; ♣8— ♡7;♠J; ♣7; ♡K; ♡8; ♡A;♢7; ♡Q; ♡J; ♣9;
5. ♠9;♡K; ♡10; ♢Q; ♢8; ♣K;♣8; ♡8; ♢J; ♡9— ♠10;♡7; ♣J; ♠7; ♠8; ♣7;♢9; ♠A; ♣10; ♢7— ♡J;♢A; ♣9; ♡A; ♠Q; ♢10;♣A; ♡Q; ♢K; ♠K;

6. ♡Q;♣A; ♢Q; ♡A; ♠J; ♠10;♢9; ♢8; ♣J; ♢7— ♡K;♣9; ♢J; ♠8; ♠7; ♠9;♡7; ♡9; ♣Q; ♣7— ♣8;♢A; ♣10; ♠Q; ♠K; ♠A;♢K; ♢10; ♡J; ♡8;
7. ♢10;♣K; ♠8; ♠K; ♣9; ♡J;♡10; ♢K; ♣J; ♠A— ♡Q;♡K; ♣8; ♢J; ♡A; ♢9;♡9; ♡8; ♣A; ♣Q— ♡7;♢7; ♠10; ♢A; ♠9; ♠7;♣10; ♠J; ♣7; ♠Q;
8. ♠K;♢K; ♣K; ♡Q; ♡9; ♡7;♣A; ♠10; ♡J; ♣10— ♡8;♣J; ♠A; ♠Q; ♡10; ♢9;♠8; ♣8; ♠J; ♢10— ♢Q;♢7; ♢A; ♣7; ♠7; ♣Q;♠9; ♢J; ♢8; ♣9;