24th European Conference on Artificial Intelligence - ECAI 2020
Santiago de Compostela, Spain

Representing and Reducing Uncertainty for Enumerating the Belief Space to Improve Endgame Play in Skat

Stefan Edelkamp
Faculty of Computer Science
University of Koblenz-Landau
stefan.edelkamp@gmail.com

Abstract.

In most fully observable board games, current AIs outperform expert play. For partially observable trick-taking card games, however, human experts still play consistently better. This paper proposes efficient knowledge representation and reasoning algorithms for the internationally played three-player card game Skat by representing, progressing, enumerating, evaluating and voting for the possible *worlds*, each player refers to as his/her knowledge about the other players' and the Skat cards. By using expert rules, elicited from statistical information in millions of games, this knowledge is accumulated in the first few tricks in order to reduce the uncertainty in the players' belief. In the so-called endgame, after five to six rounds of trick play, refined exploration algorithms suggest cards that lead to improved play. The proposed AIs have been tested both in reconsidering recorded human games, and in interactive play.

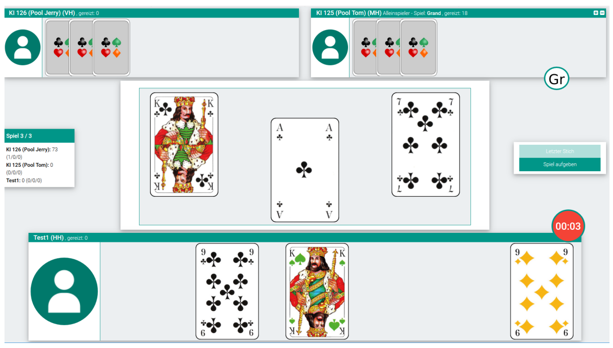
KI 126 (Pool Jerry) (VH) , gereizt: 0
KI 125 (Pool Tom) (MH) Alleinspieler - Spiel: Grand , gereizt: 18
Gr
Spiel 3 / 3
KI 126 (Pool Jerry): 73 (1/0/0)
KI 125 (Pool Tom): 0 (0/0/0)
Test1: 0 (0/0/0)
Letzter Stich
Spiel aufgeben
00:03
Test1 (HH) , gereizt: 0

Figure 1. Skat played on our Webserver against two AI clients.

1 Introduction

After many fully observable board games like Chess, Go, and Shogi [25, 26], and some trick-free card games like Poker have either been solved or AIs play beyond human strength [1], there is an increasing number of approaches to tackle multi-player trick-taking card games with randomness in the deal and incomplete partially-observable information, especially in Skat, often employing advanced machine learning methods [18, 14, 7, 21, 20, 6, 27, 12, 6, 3]. According to a professional Skat player we work with, despite the gap steadily closing, human experts still play consistently better than computer engines. Skat itself (see Figure 1) is an internationally played trick-taking card game [15, 9], which currently attracts researchers, searching for a representative of one of the next big AI challenges. The national card game of Germany and one of the most popular one played in Poland and France is widely considered the most interesting card game for three players. Skat is played with a deck of 32 cards. After shuffling, each player gets a *hand* of 10 cards with 2 cards remaining on the table, the *Skat*. After the initial stages of the game, *bidding* (to determine one declarer and two of his opponents), *Skat taking* and *Skat putting* (optional), the crucial aspect in the *trick-taking* stage is reasoning about and reducing the uncertainty in the remaining cards that are not visible to the players.

The *belief space* of remaining *worlds* of possible hands for the other two players and the Skat can be large, so that this *information set* is usually sparsely sampled and evaluated using an open card game solver [18, 14, 7]. Recent work suggests that drawing the sample wrt. a non-uniform distribution leads to better results than sampling from a uniform distributions [21]. Trick-taking, as in Skat, leads to considerably long histories, to which refined learning algorithms for partial information games like the hot topic *counterfactual regret minimization* [2] hardly scale [20, 27].

Perfect information Monte-Carlo sampling (PIMC) introduced by Levy [17] is widely considered to be the best algorithmic options for dealing with such imperfect information games. It was already been used in Ginsberg's popular Bridge-playing program GIB [8], and taken on to other trick-taking games like Skat [14, 7], or Spades/-Hearts [28]. An analysis of PIMC is given by Long [18]. The algorithmic take in PIMC is, at each decision point to select a card, to evaluate a larger sample of the belief space and call a double-dummy solver for each of the worlds, followed by selecting the card with maximum score. Furtak [7] has proposed *recursive Monte-Carlo search* to improve PIMC. Some limitations have been identified for Bridge play as matters of *strategy fusion* and *non-locality* by [5], leading to the $\alpha\mu$ search algorithms. The main observation is that even if the full belief space would be sampled and analyzed, the individual searches in PIMC may lead to contradicting card proposals. The main contribution of $\alpha\mu$ is to increase the lookahead (parameter M) in PIMC for a better exploration/exploitation trade-off. The increase in running time is reduced by further pruning rules. In its nestedness the recursive strategy shares similarities with *nested Monte-Carlo search* [4, 30] and *nested rollout policy adaptation* [22].

In this paper we expand the reasoning for reducing uncertainty in the endgame, which we consider to start around the fifth or sixth round of play. During the opening and middle game, other player' cards are either included into or excluded from the belief. The smaller the number of *worlds*, the faster and more accurate a complete exploration and evaluation of the entire belief space, for which we employ a confidence voting scheme to determine the next card to be played. With an open-card solver we evaluate up to a thousand different worlds for a card to be chosen, with only a minor drop in the performance. To ease play and to reduce the uncertainty in the belief with *knowledge moves* the two opponent try to exchange as much of extra information on their cards as possible. As the three players all have different hands and beliefs, of course, the possible worlds they are facing vary among them.

The main contributions of the paper are as follows.

1. In our set of selected we found the human expert declarer to win with a probability of 75.3% (about 83% are common in Suit games, with Grand games won by about 93%). The open card solver (often called double-dummy [14]) for the declarer on the same input wins significantly fewer games, which indicates that Skat is a game sensible to incomplete information, many games are won based on the uncertainty of the opponent players!
2. We solved the problem of many different worlds in the belief space to determine an (almost) optimal playing card. After Skat taking these are 184,756 worlds in the view of the declarer, and 42,678,636 worlds in the view of each opponent. Even for an engineered version of our worst-case solver, for a real game the time taken was infeasible, as in interactive play our Skat server demands at most 1.5 minutes for the selection of a card.
3. We devised another concept to cope with uncertain information.
   (a) choice of game, Skat putting and close-to optimal opening rely on expert rules and statistical information for both parties.
   (b) the first few tricks are played following expert rules with the goal to clarify the distribution of cards, to reduce the number of worlds in the belief space, and to weaken the opponent.
   (c) all distributions of cards are determined that are consistent with the current belief in the other players' cards and in the cards put into the Skat.
   (d) for up to 1,000 worlds the best possible card to be played is determined through a voting scheme on the results of an open card game solver.

In addition to [6], who concentrates on bidding and putting strategies and Null(Ouvert) play, this paper advocates advanced Trump play, evaluating the gains of a belief-space player for the endgame.

The paper is structured as follows. We kick off with a brief look on the Skat rules for Trump games, followed by the way to extract knowledge contained in a database of millions of expert games for determining probabilities of winning, the suggestion of first cards, and neglecting Skats from the opponents' belief. Next, based on statistical information extracted from the database, we show how to provide card recommendations that are provided to the players in form of opening tables. A rudimentary reactive player is discussed, from whom we refined its implementation. We discuss the impact of playing conventions to transfer knowledge between opponent partners and have a look at possible implementation for deciding the game that lead to modeling and continuously updating the uncertainty in form of knowledge vectors, which, in turn, are used to count, generate explore, and vote on every world in the belief of each player. The empirical results indicate the positive effect of endgame play in increasing the winning ratios of the players; an effect which we study more closely by varying its parameters.

2 Skat Rules for Trump Games

The game of Skat has been well studied in the literature [10, 11, 16, 13, 19, 23, 24, 29], and goes back to Johann Friedrich Ludwig Hempel in the year 1848 [9]. Competitive play is ruled 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 4 suits (♣, ♠, ♡, ♢). After shuffling, each player receives 10 cards, while the Skat consists of 2 cards. There are four stages of the game: the bidding stage, taking and putting the Skat, and the actual play for tricks. The *declarer*, who has won the bidding, plays against the remaining two *opponents*, while usually being able to strengthen his/her hand by taking the Skat and putting it (can be the same ones).

Bidding usually follows a predefined order of calling subsequent *values* of the game $(18, 20, 22, 23, 24, 27, 30, 33, 35, \ldots)$. Human players frequently use a bidding strategy that hides what they would have been able to bid for, by not bidding up to this value, jumping, or sometimes going a step or two further. The bidding value of Suit and Grand games depends on the distribution of Jacks, The final bids indicate in which suit players are strong, so that besides the value itself we keep 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. Per default the game with highest expected payoff is chosen. In our implementation, however, we apply a *dynamic bidding* strategy, being able to change the game aimed at that respects the current bid.

Card values in Trump games are added for the tricks being made and the Skat being put, with a usual split at 60 of the 120 possible points. Other contracts (89 – Schneider, and 119 – Schwarz) are possible. Announcing higher contracts yields a better bidding values but is only available if *Hand* is played (Skat not taken). The bidding value depends on the distribution of Jacks: As the multiplier of the suit's value ($12 = $ ♣, $11 = $ ♠, $10 = $ ♡, or $9 = $ ♢) or 24 (Grand), 1 is added to the number of consecutive Jacks in the order ♣, ♠, ♡, and ♢; or the number of consecutive Jacks in the joint hands of the opponents. If high Jacks are found in the Skat during Skat-taking, or, in Hand games, at the end of the trick play, bidding values may drop, and in the worst case lead to loosing a game that fulfills the contract.

Work on an efficient open card solver [14] went into an expert-level Skat player [18]. Symmetries and refined search algorithms have been studied in [7]. Besides scientific developments, there are commercial Skat apps like *Skat* (https://www.skat-spiel.de) of rising strength, where, due to the lack of public information, it is not entirely clear, if the AIs look into the human player hands.

3 Winning Probabilities and Ignoring Bad Skats

For the design of top computer card game players processing a huge set of human games is essential in order to elicit statistical knowledge and, in turn, to apply machine learning algorithms. For this to happen, we process a database of tens of millions of expert games. The ID of the game, the hands and Skat in the deal, and the type and contract of the game played, the maximal bidding values, the human Skat, the tricks of human play, and the according score are recorded.

For Trump games well-selected features help to accurately predict the probability $P_w(h)$ of winning a hand, which, in turn is the important ingredient for bidding, for the declarer to decide on the game being played after Skat taking, and to select two cards to be put [6].

Given $3\binom{32}{10,10,10,2}$ different Skat deals (including the turn) storing a lookup table for $P^t_w(h)$ 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. In fact, in the bidding stage for each of the games t played and current bid, we enumerate all possible $231 \cdot 66$ Skats, while ruling out some impractical ones.

In his book, Gößl [9] identified relevant winning features for Trump games: 9 for Suit games, and 7 for Grand games, ranging from the distribution of Jacks and the number of Non-Trump Aces and Tens, the number of Trump cards, via the number of free suits, and the empirically determined number of cards standing in each suit, to the value put into the Skat and the bidding value of each player.

Based on the rich corpus of games, Edelkamp [6] showed that using a hash table perfectly addressed with the above winning features, the winning probability $P_w^t(h)$ of expert play for a hand h and a given game of type t can be approximated within 3% of accuracy. Together with an equally precise winning probability derived for Null and Null-Ouvert Games and variants for Hand, Schneider and Schwarz games, this value can be used to compare the expected payoff for different game choices during the bidding stage, the selection of the game, once the Skat is taken, and the most promising cards to put into the Skat. For the proper retrieval of P_w^t in a distributed playing scenario, as demanded by the server, we secured a thread-safe implementation, for which we chose a plain static array together with a binary search for finding the unique hash address. We also used the winning probabilities $P_w^t(h)$ to separate the *good* (likely) from the *bad* (unlikely) Skats to reduce the number of worlds of the belief space. As we assume expert players, in the opponents belief on the way the declarer puts, we simply ignore Skats that are $> 20\%$ off the winning probability for the best assessed one.

4 Recommendations based on Opening Tables

Next, we extract information from the expert games for proposing the first cards to issue, and cards recommended to take over (cut) the trick, together with a suggestion on how to continue on playing.

Some information about the card distribution can be extracted already from the bidding stage, which via decoding the bid values, indicates in which suits the opponent partner is strong, urging the players to prefer one suit over the other. For starting trick-taking play, however, the knowledge of the distribution of the cards is still rather weak. For further improved opening stage of the game, we have analyzed millions of expert games, and distilled playing information in tables for the both the declarer and the opponents.

In Suit games, for example, there are $2^{11} = 2,048$ possible patterns for Trump cards used for constant-time table address. (The actual table only consider 5–7 Trump cards and has 1,254 entries.) To reduce the number of possible Skats further, we generally assume that no Trump card is put into the Skat. For Grand games with at most four Trump cards the tables with $2^4 = 16$ entries are of course smaller, and the detailed decision procedures when to play Trump or Suit differ. Nonetheless, modern Skat theory, as reflected in our program, unifies both types of Trump play. We have first compiled a small table that roughly suggests card groups (low/high-value trump, small/big jack, non-trump) for the first, a cutting, and a follow-up card, but then we noticed that we could distill more precise table information, suggesting several exact cards to be played in the opening. Once we know that a Trump card has to be played, another perfect hash function is used to recommend a proper one.

For the declarer D the first two perfectly addressed tables D_1^I, D_2^I denote, which Trump card to issue (I) first (1), and which card to issue second (2). The reasoning behind the decision, whether to issue a Trump or a Non-Trump card is complex and includes not only the number of remaining Trump cards, but also considerations, if the hand of the declarer is *trump-strong*, with more than 7 Trump cards, or if it is *trump-weak* with less than 5 Trump cards.

Similar to the tables for card-issuing, by analyzing the expert games, we generated two perfectly addressable tables for cutting D_1^C, D_2^C denoting the Trump card to overtake the trick, once the issued card on the table cannot be obeyed. The declarer's decision whether to take over, or to let go, relies on rather involved criteria, including the position in the trick and the points that are on the table.

If a non-Trump card is chosen by the declarer, then the suit to choose the next card from is essential, as some suits provide better long-term opportunities e.g., by manifesting *standing cards*. As there are 7 non-Trumps in each suit, we use priorities D^O in $\{1, \ldots, 10\}$ for each distribution on which one has best outcome, and, once a suit has been selected, which card D^S in $\{A, T, K, Q, 9, 8, 7\}$ within this suit to choose.

We also have compiled opening tables for the opponents, e.g,, to prioritize which non-trump suit to select first, and to return the card within the chosen suit offers the highest probability of winning. For the opponents we additionally store with each priority of a suit to be issued, a probability in case he plays the Ace or prefers a card below (so-called Under-Ace play [9]). There are many other *sins* in Skat, that must be avoided to concentrate the play and maximize the knowledge transfer between the players, like replaying the suit that has just been played by the partner, not opening a third non-trump suit, adjusting the play according to the bidding value, etc. [9].

5 Reactive and Advanced Player

Together with the opening card recommendation it is possible to devise a very first reactive AI player, which was the basis for later development of players that covered more strategic play, up to advanced techniques like a) showing the *antisuit*, a strong weapon in Grandplay, b) dropping a card in a suit mainly to indicate that the suit that is paired up with, should be played by the opponent partner as he has an Ace, or c) the *red-jack rule* that prefers playing ♡J in case of having ♢J, to indicate to the opponent that s/he and not the declarer has it [9], and more refined card dropping strategies.

To leave the reader with a very rough impression on the flow of reactive play, we exemplify such naive player for the declarer in forehand position.

- if non-trump-card playing criterion is met,
  - if non-trump hand card present in hand,
    - ∗ if recommendation on non-trump card exists, return it
    - ∗ else return high-value non-trump card
  - else return high-value trump card
- else // criterion not met
  - if low-trump game, return high-card
  - if high-trump game ,return low-card
  - if recommendation on trump card exists, return it
  - return low-value card

The naive player *declarer-middlehand* roughly looks as follows.

```
• if high or low-trump game is played,
  – if trump card is issued,
    ∗ if trump card present in hand,
      · if issued card higher than hand cards, return low trump card
      · else return high trump card
• else // no trump played, or standard game
  – if issued suit in hand, return obey-suit
  – else return take-or-drop
• if trump card is issued
  – if trump card present in hand,
    ∗ if issued card higher than hand cards, return low trump card,
  return high-value trump card
• else // no trump card is issue
  – if issued card can be be cut,
    ∗ if low-card played and singleton exist, return it
    ∗ if recommendation card exists, return it
    ∗ if issued trump card higher than all hand cards, return low
      trump
    ∗ if trump card present in hand, return high-trump card, else
      return low-value card
  – else return has-to-obey
  return low-value card
```

The pseudo-codes of the functions *declarer-rearhand*, *opponent-forehand*, *opponent-middlehand*, *opponent-rear-hand* are similar in spirit. Wrt. the basic player, several expert rule refinements have been implemented since, e.g., to prefer or secure high-value cards (10s or Aces), to immediately return the suit an opponent partner has played, and to take care if the number of trumps is getting small.

6 Knowledge Exploitation and Transfer

Due to the lack of information about the cards of the other players and the Skat, the belief-space is defined as the number fully observable *worlds* that are consistent with his/her knowledge, and a player might be truly facing.

As cards that have been played are removed from the players' hand, the amount of uncertainty in the number of possible *hands* shrinks naturally over time, but —depending on the cards being played— more information on the other players' hands and the Skat can be derived, especially if the players collaborate in following playing conventions, and prefer playing information-providing cards that reduce the number of possible hands and Skats.

Especially for opponent play, there are expert information-conveying rules like

> if declarer leads the trick, select lowest-valued card; otherwise choose the one with the highest value.

The point is that by knowing that the partner's card is highest (or lowest), one can exclude all cards, that are higher than the played one from the belief. This information is used to strengthen the knowledge by memorizing cards that another player must not have.

This belief, as maintained in the exploration, is sometimes characterized as the *information set*. There are many rules to decrease the sizes of the information sets significantly. One of the strongest option for reducing the degree of uncertainty is available via looking at the cards being played when a suit is not being obeyed.

7 Worst-Case Belief-Space Search

An efficient open card solver has been proposed by Edelkamp [6]. Another fast *double-dummy* solver has been contributed by Kupferschmid and Helmert [14]. Therefore, in this section we concentrate on presenting a worst-case belief-space search algorithm. As with the High-Card Theorem for Grandplay [6] for all three players it is important to infer, when the game is decided, i.e., won by certain, regardless of the remaining uncertainty of the game. In other words, the game is won in all possible worlds of the belief space, and suggests what is called *strong* play.

As the belief-space may only be approximated, (e.g., by the assumption on a good Skat putting) one may not be able to fold the game directly, but use the proposed cards of this exploration as an autopilot. For this worst-case analysis, there are basically two different implementations.

7.1 Iterated Open-Card Search

The first approach is to generate all the possible distributions compatible with the current belief, and to subsequently start the open card solver (alias *glassbox*), to decide, whether or not all individual games are won (or lost). They all have to agree on a game-deciding card.

The glassbox refers to an efficient algorithm in the form of a And-Or tree search. We exemplify our considerations for the declarer's perspective at the beginning of the game with 20 unknown cards, we have $20!/(10!10!) = 184,756$ different distributions, of which only one is true. For the opponents who additionally don't know the Skat there are $22!(10!10!2!) > 42M$ possible card distributions.

For the start of the game, it is, therefore, impossible to analyze all possible words in the decision time of up to 90s. If the card set is reduced to say 5 cards, however, the search is much faster, due to the number of world shrinking exponentially to $10!/(5! \cdot 5!) = 252$ (declarer's view). And, of course, there is also the acquired knowledge about the distribution of the cards resulting from the play and can greatly reduce the number of possible worlds.

We will use this approach later on to vote on the best card in the endgame.

7.2 Paranoia Search

Another analysis is to unify the search tree and the possible card distributions into a single worst-case analysis. As a first insight, the many individual analyzes in iterated search can be cast as a single root choice node.

In the following Paranoia analysis, the declarer plays against all possible hands of opponent cards simultaneously. For example, if Opponent 1 does not obey one suit, (alternatively, Trump), all cards of that suit in the part of the search tree will automatically be transmitted to Opponent 2 (and vice versa). Players must first obey the issued card with the already permanently ones assigned to them, before they can choose a free card from the pile of remaining ones.

The concept of this Paranoia analysis of the declarer is tricky to understand because the search tree of card choices is interleaved with the choice points for hidden cards, so that the entire tree is solving a game of uncertainty.

First of all, the procedure (shown in Figure 2) needs access to the hand, of course of the declarer (`hands[0]`), the Skat, the position within the trick, the points scored on both sides so far, etc. As the declarer knows its hand, for his *And* search node in the And-Or search tree, nothing changes to the open card analysis.

If we assume deciding the entire game, and do not include information conveyed in the bidding, the declarer has no prior knowledge about the current distribution of the opponent cards at the beginning. S/he initially only knows the set of 20 cards, but not the individual hands. All cards are managed in one vector `both`, initialized with all 20 cards of the opponents. Their hands are initially empty. We monitor, e.g., when one opponent player is not obeying a suit, which allows moving all unknown cards to the other opponent's hand. The currently certain knowledge about the cards in the search tree is stored in vectors (here `belief[1..2]`).

The cards assigned to the one opponent players are no longer available to the other opponents. One important observation is that no more than 10 cards can be shifted to a one opponent player. In that case, the movement of cards must be stopped, pruning the search tree. This avoids that a player suddenly serves a suit, which he has to discard. In other words: we have significantly reduced the level of uncertainty, since we know, where some cards are located.

The beliefs are updated within the search tree. We carry out a search in such mixed search tree of a) cards being played and b) inferences being enforced. In the tree, we have to test all possibilities consistent with the knowledge at the node, so the tree grows quickly. If during search, an opponent has to obey an issued card with a card assigned to him/her, no longer there is a free choice from the other unassigned cards in the current turn.

Implementing the mixed search was far more involved than programming an open card solver, as it is not only about progressing the vectors `hands[0]`, `belief[1]` and `belief[2]` in the nodes of the tree search, but also to perform a proper restoration from the recursion in case of backtracking. This includes updating the score (stored in `as` for the declarer and in `as` for the opponents). For the opponents knowledge vectors might have to be unified.

Two subtleties. Open-card and Paranoid solvers avoid searching previously considered states. A *transposition table* stores states with their analysis result. Without the transposition table, search becomes inefficient. We encountered that the existing encoding of a state in open-card play required only 32 bits of an unsigned int, while for Paranoia search called for a unsigned long.

We tested the Paranoia search engine. While it worked and was much faster than the enumeration of open card solver calls, at the end of the day, however, we could decide only very few simple full games in reasonable time. By looking at the even larger belief-space for the opponent, we neglected Paranoia search from our player. While a worst-case solver helps to solve puzzles, during play, we aim at collect guiding information on good cards, not only on perfect, 100% winning cards.

Nonetheless, the algorithm design and ideas on knowledge progression in the search tree, guided us to the model of uncertainty that we chose for the players.

8 Modeling Uncertainty

While players in the Skat game can see the cards on the table, based on their knowledge of their own hand, they all have to draw different individual conclusions on the remaining set of possible or at least very likely card distributions. To model the uncertainty in the Skat game, we store and update *knowledge vectors* (32-bit unsigned integers), each of which denotes a set of cards believed *not* to be present at another players' disposal. For example, if one player fails to obey the suit of the issued card, either by dropping a card or cutting it, we know for certain that none of the remaining cards in the suit can reside on the players' hand.

```
OR1(playable)
 if (cardontable)
   reduced = reduce(playable & ∼belief[2]);
 while (reduced)
   s = select(playable);
   b = belief[2];
   if (obey(s))
     if (!playable(hands[1],index,1))
       playable &= ∼s; continue;
   else
     if (trump(cardontable))
       if (!(trump(s)))
         hands[2] |= trump(both);
         if (|belief[2]| < 10)
           belief[2] = b; playable &= ∼s; continue;
     else
       if (!(suit(cardontable,s))
         belief[2] |= suit(c) & ∼trump(both);
         if (|belief[2]| > 10)
           belief[2] = b; playable &= ∼s; continue;
   both &= ∼s; hands[1] |= s;
   if (|hands[1]| > 10)
     belief[2] = b; hands[1] &= ∼s;
     playable &= ∼s; continue;
   played |= s; t[1] = s;
   if (trickdone)
     turn = winner(2,0,1);
     if (turn) gs +=score(i); else as += score(i);
     t0 = t[0], t2 = t[2]; t[0] = t[1] = t[2] = -1;
     rval = (gs >= 120-LIMIT) ? 0 : (as > LIMIT) ? 1 :
       (turn == 0) ? AND(hands[0]) :
       (turn == 1) ? OR1(both) : OR2(both);
     t[0] = t0; t[2] = t2;
     if (turn) gs -=score; else as -= score;
   else rval = OR2(both);
   t[1] = -1; both |= s; played &= ∼s;
   belief[2] = b; hands[1] &= ∼s; playable &= ∼s;
   if (rval == 0)  return 0;
 return 1;
```

Figure 2. Worst-case tree search, opponents move, declarer's view.

First, for each player we maintain and update knowledge vectors A_i, $i \in \{0, 1, 2\}$, denoting all cards that have not being played and that can be still assigned to the other two players' hand or the Skat. This vector A_i is initialized with the current hand of the player.

Next, we have knowledge vectors split into four vectors for each of the players 0, 1, 2 and the Skat. The forbidden cards kept in knowledge vectors are denoted by F_i^j keeping track of the knowledge in the view of player $i \in \{0, 1, 2\}$ on player j with $j \in \{0, 1, 2, Skat\}$, $i \neq j$ for not having a card. For example F_2^1 contains all the cards that Player 1 knows Player 2 cannot have. For the declarers view on the Skat (at least for games that are not Hand) we initialize the vector F_0^{Skat} with the other 30 cards.

8.1 Counting and Evaluating the Belief-Space

For mapping the knowledge, succinctly encoded in the knowledge vectors, to the belief-space of possible hands and Skats, we use a constructive approach. For Player i, takes all three vectors of forbidden cards F_i^j together with A_i (i.e., the cards to distribute) and recursively generates all possible worlds. More precisely, we use two different almost matching function, one to count the number of possible worlds, and one to generate them to subsequently apply the open card solver. Counting allows to set a threshold to the maximal number of worlds in the belief space for the open card analysis, without actually starting the solver. By the speed of modern computers, hundreds of thousands of worlds can be generated and counted in less than 1s.

The recursive pseudo code for generating all possible cards wrt. a set of knowledge card vectors known not to be at the other play-

```
enum1(r,h1,h2,sk,not-h1,not-h2,not-skat)
  if (|r| == 0)
    sk1 = select(sk)
    sk2 = select(sk & ~sk1)
    if (!declarer)
      p = winprob(h1,sk1,sk2)
      if (p >= best) best = p
      if (p >= thresh) solve(h1,sk1,sk2)
    else solve(h1,sk1,sk2)
    return 1;
  c = select(r)
  if (|h1| < 10-r && !(c & not-h1))
    enum1(r&~c,h1|c,h2,sk,not-h1,not-h2,not-skat)
  if (|sk| < 2 && !(c & not-skat))
    enum1(r&~c,h1,h2,sk|c,not-h1,not-h2,not-skat)
  return 0

enum(m,r,h1,h2,sk,not-h1,not-h2,not-skat)
  if (|r| == 0)
    best = thresh = 0
    enum1(rem,h1,h2,sk,not-h1,not-h2,not-skat)
    thresh = best - best * p/100
    enum1(m,h1,h2,sk,not-h1,not-h2,not-skat)
    return 0
  c = select(r)
  if ((|h2|<10-t) && !(c & not-h2))
    enum(m,r&~c,h1,h2|c,sk,not-h1,not-h2,not-skat)
  if (|m| < 12-t)
    enum(m|c,r&~c,h1,h2,sk,not-h1,not-h2,not-skat)
  return 0

hands(all,not-h1,not-h2,not-skat)
  h1 = h2 = sk = 0;
  enum(0,all,h1,h2,sk,not-h1,not-h2,not-skat)
```

Figure 3. Enumerating information set of one player.

ers hand or in the Skat is shown in Fig. 3. The procedure mainly distributes the cards in A_i into the hands and the Skat, in an enumerating backtrack procedure of in- and exclusion until the pool of cards runs empty, in which case one of all possible distribution of cards (a world) is established.

The code itself chooses the next undecided card in either of the hands or in the Skat, while satisfying the imposed constraints. The advanced part of the procedure is to organize the enumeration of worlds in a way that keep same Skats together, which in turn is needed to determine the best one and to limit the deviation in its winning probability. This option allows opponents to determine good Skats and neglect bad ones.

Therefore, in the algorithms of Fig. 3 we discard distributions with bad Skat-putting that we expect a reasonably well-playing declarer not to have put. We use the winning probability function that for this and discard ones that are of by $p\%$ from the best one (e.g., $p = 20$). For this we retrieve the best possible Skat putting for a given distribution of cards to the hands. To enumerate all possible Skats for each hand together, the recursion is two-staged and first split wrt. the declarers' cards, and then to distribute wrt. the skat.

Another complexity that is not reflected in the pseudo- but present in the real code is the recommendation of cards for tricks with cards on the table, inducing an unbalanced number of cards in the procedure.

8.2 Voting with Confidence

Once all worlds have been generated, pruned and evaluated with the open card solver, proposing at least one valid playing card, we start voting on the best one applicable to most worlds. We use a simple vector B of size 32 initialized to zero, and increase the value at position i if the card index that has been proposed has value i (in one suitable encoding). In a second loop we determine the one that has been given the maximum number of votes, i.e., the card i with maximum value $\max_{i=1,\dots,32} B_i$. With the total amount of votes V, which equals to the number of worlds in the belief space of the player, we can compute the confidence C in the decision, defined as the ratio $C = (\max_{i=1,\dots,32} B_i)/V$. In the experiments we impose different thresholds on C. The higher the threshold, the stronger the confidence required to accept the recommendation of the belief-space analysis. If the value of C is too small for exceeding the threshold, we resort to the default recommendation of the expert rules, which is always applicable.

9 Experimental Evaluation

We implemented our Skat AI in C++ (gcc, version 7.4.0 Ubuntu-8.04.1). Depending on the makefile chosen, it features interactive play on the server, or database play from a file of recorded expert games. The players itself run in (p)threads and are connected via a base (client-pool) API to the remote server.

The implementation uses one player class for bidding, Skat putting, and Skat taking, and virtual functions for trick-taking play, covering the entire taxonomy of games to be played: a base class for all types of play, and derived ones for Null and Null-Ouvert and Trump games, with the latter being used as a super class for Grand and Suit games, on which we stress our attention.

In the server GUI (see Fig. 1), which runs in any Internet browser, the user can choose among different AI client-pools, and let them play against each other or against humans that are logged in. Interacting with the server, the AI players support all stages of the game. All AI games are recorded and can be used for a detailed analysis including forensic replay. One default AI chooses random but valid cards. It serves as a dummy C++ player implementation for the development of an own Skat engine, suitable for a face-to-face comparison in an AI on-line Skat tournament.

Using a C/C++ library for web-sockets and a textual JSON protocol exchange format, we have attached our AI players as clients to an existing server for human players. To avoid concurrency problems, in the client all global information is static. We can conveniently add up to three different AIs (of the same or different client-pool), playing against each other. In a series of 1,000 games, played on the server in 12m22s two default AIs scored 47 (wins) to 235 (losses) with a score -15,411, and 49 (wins) 276 (losses) with a score of -20,818, while our AI won 317 games and lost 19 games with a score of 50,595. A professional player participating in World, European and National championships frequently wins against the AI by a large margin, but managed to loose at least one series of 12 games (validating that Skat is a game of chance).

Evaluations on randomly chosen games leads to rather inconclusive evaluations, as some hands clearly feature the one game selection, and not the other (simply compare playing cards strong for Grand with strong playing cards for Null). For the evaluation we, therefore, extracted 60,000 expert games of all kinds from our database and took the result of the human bidding stage to select the type of game. For a clear comparison, we looked at Trump especially Suit games (no Hand nor Ouvert). The AIs select the same game that was chosen by the expert declarer. We also selected the Skat that was put by the human declarer. We tested, whether or not endgame recommendations by enumerating and evaluating the belief space, helps the players. The computer runs an Intel(R) Core(TM) i7-7600U CPU @ 2.8 GHz and has 16 GB RAM.

We started with an initial series of 1,000 Suit games (of the 60,000) and found that the human declarer won 800, the glassbox solver only

217, and the AI 826. With endgame support for the declarer the last value increased to 838, and with opponent endgame support dropped to 793. While the margin appears small on the first glance, by the level of play, it is significant.

In the following, we analyze the findings in more detail. For 10,000 Suit games Table 1 shows the results obtained, partitioned along the possible outcomes of human and glassbox solver play. We varied the setting with the declarer and/or the opponent using endgame support. The CPU time for the analysis in all settings we examined, varied between 10m and 14m, so that each individual game was played in less than 0.1s. In the table we can see that endgame support for the declarer pays off, increasing the number of games won. Similarly, the endgame support of opponents decreases the number of declarer victories.

Human	Glassbox	AI	D^+O^-	D^-O^+	D^-O^-	D^+O^+
loses	looses	wins	1,132	1,075	1,095	**1,157**
loses	wins	wins	**134**	128	131	129
wins	loses	wins	**5,054**	4816	4984	4,979
wins	wins	wins	1990	1952	**1,991**	1,961
*	*	wins	**8,310**	7,871	8,301	8,226

Table 1. Declarer wins of playing 10,000 Suit games with (+)/without(-) belief-space suggestion for declarer (D) and opponent (O). The human declarer determined the two cards for the Skat.

If instead, we look at the number of games won by the human expert, the AI declarer establishes significantly more wins: 8,310 vs. 7,999. This is an indication of considerably good play of the AI, but does not necessarily mean that the AI declarer is playing superior to the human one, as the result could at least partly be assigned to the case that human opponents are playing stronger than the AI ones.

Next, we analyze the effect of changing the confidence, namely the minimum percentage of card proposals have to be best. In Table 2 we see that requesting a higher confidence level leads to better results. In fact, insisting on a matching card recommendation in all worlds of the belief space led to the best result.

Human	Glassbox	AI	0.1	0.4	0.8	1.0
loses	looses	wins	1,130	1,132	1,172	**1,176**
loses	wins	wins	134	134	**135**	132
wins	loses	wins	5,052	5,054	5,102	**5,103**
wins	wins	wins	1,990	1,990	2,011	**2,014**
*	*	*	8,306	8,310	8,420	**8,425**

Table 2. Declarer wins of playing 10,000 Suit games with belief-space suggestion for declarer, a belief-space size of $< 1,000$ and varying confidence level. The human expert declarer selected the Skat.

In Table 3 we also tested the influence of changing the maximum size of the belief space. We expected that large sizes of the belief space do not change the outcome significantly, as the number of belief states after 5 rounds of trick play, especially for the declarer, is often small. To our surprise, a very smaller belief space of 10 led to even better results. This is likely due to more concentrated card proposals.

Table 4 shows the result of varying the number of tricks, afters which the endgame assistance is invoked. We see a sweet spot at about round 5 and 6.

For finding card recommendations automatically we chose an

Human	Glassbox	AI	10	100	1000	10000
loses	looses	wins	**1,146**	1,132	1,132	1,132
loses	wins	wins	**135**	**135**	134	134
wins	loses	wins	5,033	**5,054**	**5,054**	**5,054**
wins	wins	wins	**1,999**	1,990	1,990	1,990
*	*	wins	**8,313**	8,311	8,310	8,310

Table 3. Declarer wins of playing 10,000 games with belief-space suggestion for declarer, a required confidence of $> 40\%$ and varying size of the belief-space. The human expert declarer selected the two cards for the Skat.

Human	Glassbox	AI	4	5	6	7
loses	looses	wins	1,096	1,132	**1,145**	1,108
loses	wins	wins	**135**	134	133	133
wins	loses	wins	5,000	5,052	**5,046**	4,991
wins	wins	wins	1,982	1,990	1,993	**1,996**
*	*	wins	8,213	8,310	**8,317**	8,228

Table 4. Declarer wins of playing 10,000 Suit games with belief-space suggestion for declarer after a varying number of trick-taking rounds, a belief-space size of $< 1,000$ and required confidence of $> 40\%$. The human expert declarer selected the two cards for the Skat.

highly engineered open-card solver that selects a card, but only based on satisfying the contract (in most cases 60 points). We also tried to find the card that gives the best possible score, utilizing a divide-and-conquer search on this threshold with mixed results: the time for analyzing the belief space went up, while the winning rate did not improve significantly.

As an evaluation summary we have that expert rule play in the opening, and endgame recommendation are in a competition for the middle game. Sometimes the one has the better card proposal, sometimes the other. In the beginning of the game the former is far better, while to the end of the game the latter becomes more prominent. Both quickly determine the card, which in many possible circumstances will be good. We see the positive effect of using endgame recommendations after a sweet spot of 5-6 tricks. If the declarer takes on a card recommendation of high confidence, the number of games won significantly increases. When varying the number of tricks to start the analysis there is a trade-off but there is a clear advantage of applying expert rules to the beginning of the game, and starting the analysis with the open card solver in the belief space to the end of the game, suggesting that the proposed architecture of the AI is appropriate.

10 Conclusion

We have seen an approach for refined belief-space search in Trump games for the game of Skat, and illustrated, how to represent the belief in form of knowledge vectors, how to refine this knowledge based while tricks are being played, and how to enumerate the resulting information to generate a set of all possible worlds for each player. While other authors sample the belief-space throughout the trick-taking stage of the Skat game, we incorporate our open card solver extensively and successfully on the *complete* belief space for the endgame, to vote on the next card to be played.

It is well-known that open-card game solving and subsequent play is fast, but it severely lacks handling the information exchange of the players, which in Skat especially for the cooperating opponents is crucial. Our proposed mix of statistical information for opening play, expert rules for handling exceptions and incorporating information-gathering moves, and endgame belief-space enumeration for deciding the game exploiting the accumulated information, pays off to greatly improve the players' performance. Confidence voting on the reduced belief space is based on experience, empirical evidence, and numerical observations. For example, if one of the 184,756 worlds a declarer might be facing after Skat-taking is evaluated in one second, it would take 51.4h to complete a series of 48 games, which is already limited to 2h of play.

There are several other trick-taking games, like Hearts, Spades, Bridge, Tarrot, that share related playing principles, but which are also different in the specifics of the games. We think that the presented approaches for bidding, trick opening and endgame play, though implemented domain-specifically, are general and can be applied to them as well, starting with changing the card encoding and obeying rules for playing the games, followed by extracting the features for winning probabilities and first cards to be played using a large corpus of expert games, together with a fast open-card game solver to decide the game especially for later stages of trick-taking.

In the server implementation we currently aim to unite database and interactive play, so that reproducible experiments become available, in the web-GUI. This improves debugging as well allow for measuring official strength values (similar to ELO in Chess). For a reliable value we will run several series being played against advanced human opponents. Unfortunately, as by today, all good Skat AIs use their own server technology and exchange format. Hence, it is hardly possible to cross-compare their strength. To overcome this hurdle, we will try organizing or participating in a Computer Olympiad in Skat. Using the server, we also plan to participate in mixed human-computer tournaments.

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

REFERENCES

[1] Michael Bowling, Neil Burch, Michael Johanson, and Oskari Tammelin, 'Heads-up limit hold'em poker is solved', *Commun. ACM*, **60**(11), 81–88, (2017).

[2] Noam Brown, Adam Lerer, Sam Gross, and Tuomas Sandholm, 'Deep counterfactual regret minimization', *CoRR*, **abs/1811.00164**, (2018).

[3] Michael Buro, Jeffrey Richard Long, Timothy Furtak, and Nathan R. Sturtevant, 'Improving state evaluation, inference, and search in trick-based card games', in *IJCAI 2009, Proceedings of the 21st International Joint Conference on Artificial Intelligence, Pasadena, California, USA, July 11-17, 2009*, pp. 1407–1413, (2009).

[4] Tristan Cazenave, 'Nested monte-carlo search', in *IJCAI 2009, Proceedings of the 21st International Joint Conference on Artificial Intelligence, Pasadena, California, USA, July 11-17, 2009*, pp. 456–461, (2009).

[5] Tristan Cazenave and Véronique Ventos, 'The $\alpha\mu$ search algorithm for the game of bridge', *CoRR*, **abs/1911.07960**, (2019).

[6] Stefan Edelkamp, 'Challenging human supremacy in Skat', in *Proceedings of the Twelfth International Symposium on Combinatorial Search, SOCS 2019, Napa, California, 16-17 July 2019*, pp. 52–60, (2019).

[7] Timothy Michael Furtak, *Symmetries and Search in Trick-Taking Card Games*, Ph.D. dissertation, University of Alberta, 2013.

[8] M. Ginsberg, 'Step toward an expert-level Bridge-playing program', in *IJCAI*, pp. 584–589, (1999).

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

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

[11] Siegfried Harmel, *Skat–Zahlen*, Klabautermann-Verlag, Pünderich (Mosel), 2016.

[12] Thomas Keller and Sebastian Kupferschmid, 'Automatic bidding for the game of Skat', in *KI*, pp. 95–102, (2008).

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

[14] Sebastian Kupferschmid and Malte Helmert, 'A Skat player based on Monte-Carlo simulation', in *Computers and Games*, pp. 135–147, (2006).

[15] Emanuel Lasker, *Das verständige Kartenspiel*, August Scherl Verlag, Berlin, 1929.

[16] Emanuel Lasker, *Strategie der Spiele – Skat*, August Scherl Verlag, Berlin, 1938.

[17] David N. L. Levy, 'The Million Pound Bridge program', in *Heuristic Programming in Artificial Intelligence*, (1989).

[18] Jeffrey Richard Long, *Search, Inference and Opponent Modelling in an Expert-Caliber Skat Player*, Ph.D. dissertation, University of Alberta, 2011.

[19] Manfred Quambusch, *Gläserne Karten – Gewinnen beim Skat*, Stomi Verlag, Schwerte Rau Verlag, Düsseldorf, 1990.

[20] Douglas Rebstock, Christopher Solinas, and Michael Buro, 'Learning policies from human data for Skat', *CoRR*, **abs/1905.10907**, (2019).

[21] Douglas Rebstock, Christopher Solinas, Michael Buro, and Nathan R. Sturtevant, 'Policy based inference in trick-taking card games', *CoRR*, **abs/1905.10911**, (2019).

[22] Christopher D Rosin, 'Nested rollout policy adaptation for Monte Carlo tree search', in *IJCAI*, pp. 649–654, (2011).

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

[24] Hermann Schubert, *Das Skatspiel im Lichte der Wahrscheinlichkeitsrechnung*, J. F. Richter, Hamburg, 1887.

[25] David Silver and Aja Huang et al., 'Mastering the game of Go with deep neural networks and tree search', *Nature*, **529**, 484, (2016).

[26] 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).

[27] Christopher Solinas, Douglas Rebstock, and Michael Buro, 'Improving search with supervised learning in trick-based card games', *CoRR*, **abs/1903.09604**, (2019).

[28] Nathan R. Sturtevant and Adam M. White, 'Feature construction for reinforcement learning in hearts', in *Computers and Games, 5th International Conference, CG 2006, Turin, Italy, May 29-31, 2006. Revised Papers*, pp. 122–134, (2006).

[29] Joseph Petrus Wergin, *Wergin on Skat and Sheepshead*, Wergin Distributing, Mc. Farland, USA, 1975.

[30] Mark HM Winands, Yngvi Björnsson, and Jahn-Takeshi Saito, 'Monte-carlo tree search solver', *Computers and Games*, **5131**, 25–36, (2008).