Knowledge-Based Paranoia Search

Stefan Edelkamp
Department of Computer Science
Faculty of Electrical Engineering
Czech Technical University in Prague

***Abstract*—This paper proposes *knowledge-based paranoia search* (KBPS) to find forced wins in the card game Skat; an internationally played card game, and likely one of the most interesting card games for three players. It combines efficient partial information game-tree search with knowledge representation and reasoning. This worst-case analysis, initiated after a small number of tricks, leads to a prioritized choice of cards. We provide variants of KBPS for the declarer and the opponents, and an approximation to find a forced win against most worlds in the belief space. Replaying thousands of expert games, our evaluation indicates that the AIs with the new algorithms perform better than humans in their play, achieving an average score of over 1,000 points in the agreed standard for evaluating Skat tournaments, the extended Seeger system.**

I. Introduction

One central showcase of artificial intelligence is to prove that computers are able to beat humans in games [1]. Success stories in playing games have been highly influential for AI research in general [2]. As many board games have either been solved [3] or AIs show superhuman performance [4], one of the next AI challenges are card games with randomness in the deal and incomplete information due to cards being hidden. While there is impressive research on playing multi-player non-cooperative card games like Poker [5], for cooperative card games like Skat and Bridge, human play appears still to be better than computer play. Despite early ground-breaking results, e.g., in Bridge [6], according to [7] best computer Bridge programs still play inferior to humans.

Another candidate for showing the intriguing challenges in trick-taking card play is Skat, an internationally played game, described by McLeod as the best and most interesting card game for three players Skat has a deck of 32 cards; in a deal each player gets 10 cards, with two left-over Skat cards. There are four stages of the Skat game: i) *bidding*, where the players communicate values towards their respective maximal bidding strength; ii) *Skat taking* and selecting the game; iii) choosing the two cards for *Skat putting*; iv) *trick-taking game play* with up to 10 rounds of play. For higher bidding values, stages ii) and iii) may be skipped. The winner of the bidding becomes the declarer, who plays against the remaining two opponents. He adds the Skat to his hand and discard any two cards. The declarer wins if he gets more than 60 points (Grand or Suit) or makes no tricks (Null). To increase the bidding value further, he can raise the contract from scoring 61 points to 90 (*Schneider*) and 120 (*Schwarz*), and also to open the hand to the opponents (Ouvert). Handling partial information is the critical aspect in this game, given that for open card play, the optimal score and the associated playing card can be found in terms of milliseconds [8].

The contribution of this paper is *knowledge-based paranoia search* (KBPS) for the trick-taking stage of the game.

The widely applied Perfect-Information Monte-Carlo Sampling (PIMC) [9], [7] may not find a paranoid strategy, even if one exists. Due to the problem of strategy fusion, it might return a card corresponding to lines of play that lack knowing the true world. Whenever a player has to play a card, we restrict the consistent worlds to only those worlds where the given card was actually legal [10]. In case of KBPS, we include knowledge representation and reasoning into the search and run the analysis for each card to be played.

We include the proposal into a Skat AI, able to play all stages of the game and all game types. Using statistical tables elicited from Human expert games, it derives accurate winning probabilities, which are used mainly for the bidding and game selection stages, and to put good Skats. For the trick-taking stage of the game, it includes algorithmic strategies for opening, middle- and endgame play using expert rules and exploiting playing conventions to build a knowledge base on plausible and effective locations of the cards. For the opening stage, winning feature statistics from expert games are stored in precomputed tables [11]. For the middle game we apply suit-factor search [12]. For the endgame the remaining space of possible beliefs is analyzed completely and recommended cards are fused using a voting scheme [13].

II. About Skat

Skat is a three-player imperfect information game played with 32 cards, a subset of the usual 52 cards Bridge deck. It shares similarities to Marias(ch) (played in Czech Republic and Slovakia) and Ulti (played in Hungary).

At the beginning of a game, each player gets 10 cards, which are hidden to the other players. The remaining two cards, called the Skat, are placed face down on the table. Each hand is played in two stages, bidding and card play.

The bidding stage determines the declarer and the two opponents: two players announce and accept increasing bids until one passes. The winner of the first bidding phase continues bidding with the third player. The successful bidder of the second bidding phase plays against the other two. The maximum bid a player can announce depends on the type of game the player wants to play and, in case of a trump game,

978-1-6654-3886-5/21/$31.00 ©2021 IEEE

Fig. 1. Skat game replay of our AI playing against a human and another AI.

a multiplication factor determined by the jacks. The soloist decides on the game to be played. Before declaring, he may pick up the Skat and then discards any two cards from his hand, face down. These cards count towards the declarer's score. An example for Skat selection is shown in Fig 1.

Card play proceeds as in Bridge, except that the trumps and card ranks are different. In grand, the four jacks are the only trumps. In suit, seven further cards of the selected suit are trumps. There are no trumps in null games. Non-trump cards are grouped into suits as in Bridge. Each card has an associated point value between 0 and 11, and in a standard trump game the declarer must score more points than the opponents to win. Null games are an exception, where the declarer wins only if he scores no trick.

III. Related Work

The game of Skat is considered in many books [14], [15], [16], [17], [18], [19]. A recent mathematical introduction to Skat playing has been given by [20]. There are frequent bachelor and master theses on the topic of Skat (e.g., by Fabian Knorr, 2018, University Passau, or by Dennis Bartschat, 2019, University of Koblenz), but due to the limited time for programming, the proposed Skat bots do not reach a human-adequate playing strength.

Kupferschmid and Helmert [21] developed the *double-dummy Skat solver* (DDSS), a fast open card Skat game solver, which found its way into the *Kermit* player [22]. DDSS was extended to cover the partial observable game using Monte-Carlo sampling [6]. It reached only moderate performance results, mainly due to lacking knowledge information exchange between the players.

There have been larger efforts to apply machine learning to predict bidding options and hand cards in Skat [23], [22], [24], [25], [26], [10]. Additionally, we have seen feature extraction in the related game of *Hearts* [27], and automated bidding improvements in the game of *Spades* [28]. The results show that the prediction accuracy can be improved.

Buro et al. [22] indicate that their player Kermit achieved expert-playing strength. A direct comparison is difficult, as the bots play on different server architectures.

Cohensius et al. [28] elaborated on an intuitive way of statistically sampling the belief space of hands (worlds) based on the knowledge inferred within play. The matrices P^i for the belief of card location for each player i show a probability $p^i_{j,k}$ for the other players j on having a card k in his hand. While the approach has been developed for *Spades*, it also applies to Skat [9]. In a different line of research, Edelkamp [11] showed how to predict winning probabilities for the early stages of the game, and how to play the Null game. Edelkamp [13]) studied Skat endgame play using a complete analysis of the belief-space that is compactly kept and updated in knowledge vectors. Referring to combinatorial game theory [29], Edelkamp [12] proposes suit factorization and mini-game search for improved middlegame play in Skat.

Sturtevant and Korf [30] described a paranoid algorithm for the case of perfect information multi-player games. In that work the player to act is paranoid with respect to the preferences of the other players, assuming that they are in a coalition against the agent. This reduces the multi-player game to a 2-player game, such that α-β pruning may be applied. Partial-information α-β paranoid search has been considered by Furtak [10], The work differs from Sturtevant and Korf's algorithm in that the agent does not have perfect information. Moreover, because the agent does not know the true world, it is also paranoid wrt. the outcome of any stochastic events (chance nodes), namely the actual distribution of any unobserved cards. The information was used for bidding and stored in tables for the $(32!/10!2!) \cdot 3 \cdot 5 \approx 224$ million hands (including game type and turn), symmetry-reduced and compressed. Edelkamp [13] considered a similar paranoia partial information search option for analyzing Skat puzzles mainly as a motivating aspect to introduce knowledge representation and reasoning in bitvector sets for endgame play. As with Furtak [10], the algorithm never went into the players' trick-taking stage, as it was cast inefficient to be useful under real-world playing constraints. In this work, we successfully integrate this analysis option into actual game play, leading to a considerable increase in playing strength.

IV. Skat Bot

Bidding and game selection both use statistical knowledge of winning ratios in human expert games, stored in tables and addressed via patterns of *winning features*. This assumes a predictor for a given hand with high accuracy, before play (no move history). We assume the player's position to be part of his hand. The winning probability of a hand decreases during the bidding stage by the anticipated larger strength of the opponent hand. Other than this, no opponent model is used.

The Skat bot estimates winning probabilities with statistical tables that are extracted from a database of millions of high-quality expert games; more precisely, winning probabilities $Prob(h, p, s, b, t)$ including current hand h, choices of game type t, Skats s, position of the player p, and bidding value b. The probabilities are then used in the first three stages of the Skat game: bidding, game selection, and Skat putting. For each bidding value and each game type selected, it generates and filters all $\binom{22}{2} = 231$ possible Skats and takes the average of the payoff of skat putting, which, in turn, is the maximum of the $\binom{12}{2} = 66$ possible skats to be put. The winning ratios in expert games can easily be analyzed statistically, but by the high number of $n = \binom{32}{10}\binom{22}{10}\binom{12}{10} \approx 2.8$ quadrillion possible Skat deals, proper generalizations are needed.

For Null games, given hand h and skat s the approach estimates the winning probability $Prob(h,s) = Prob(h,s,\clubsuit) \cdot Prob(h,s,\spadesuit) \cdot Prob(h,s,\heartsuit) \cdot Prob(h,s,\diamondsuit)$ [31], [14]. For trump games, we consult a table addressed by the so-called winning parameters: number of non-trump suits that the player lacks; number of eyes in Skat, condensed into four groups; value of the bidding stage, projected to four groups; position of the declarer in the first trick; number of trump cards in hand; number of non-trump cards in hand; constellations of jacks condensed into groups; and number of cards estimated to lose, based on summing the expected number of standing cards. Statistical tests [20] showed that these parameters have a significant influence and can, therefore, be used as essential attributes to accurately assess the probability of winning a trump game. In particular, a *Grand* table with $113,066$ entries is built on top of 7 of these winning parameters and a *Suit* table with $246,822$ entries using 9 of them. For Skat putting we refine the lookup value for different cases in a linear function together with further winning features such as the expected number of tricks while respecting the retaking options of the issuing right, and the exact number of points put into the Skat.

Trick-taking is arranged wrt. an ensemble of different card recommendations. For the sake of brevity, we refer the reader to precursor work [11], [13], [12]. In short terms, we find

- killer cards that force a win for the declarer (or the opponents) to meet (or to break) the contract of the game; this option mainly includes the KBPS card proposals of this paper; other are simpler rules that count the number of points certain to be made for the player in the remaining tricks.
- endgame cards as the results of strategy fusion, realized via a voting on the winning ratio of open card game solver calls on the remaining *worlds* in the belief space of the player [13]. The endgame player is invoked after five tricks with a maximum number of 2500 worlds in the belief space, the win ratio for a card (confidence level) is set to 90%. Additional bonus is given for a high number of eyes and for meeting higher contracts.
- hope cards as the only cards that can save the game for either the declarer or the opponents, i.e., all other cards lead to a forced loss, this card is played instantly
- expert cards for each player in each position in the trick, based on if-then-else rules that consider the current the hand of issuing players, the history of tricks being played, the partial knowledge of cards present in the opponent hands, etc.

The priority is as follows. First, killer cards are recommended; if this strategy fails to find a forced win, endgame and hope cards are searched for; if this does not meet the required criteria or confidence level, we fall back to expert cards recommendation. The expert rules, used for the first few tricks and as a default, includes card recommendations based on suit factors (either trump or non-trump). Each card in the factor is assigned a value 0, 1, or 2, where 0 denotes a hand card, 1 a card in the other players' hands, and 2 a card that is not playable (either being played or put into the Skat). For the declarer issuing trump we precomputed tables of sizes $\binom{11}{k} \cdot 2^{11-k} = 11,264$ ($k = 1$ trump), $28,160$ (2 trumps), $42,240$ (3 trumps), $42,240$ (4 trumps), $29,568$ (5 trumps), $14,784$ (6 trumps), $52,80$ (7 trumps), $1,320$ (8 trumps), 220 (9 trumps), and 22 (10 trumps). For non-trump suits a table with $\sum_{k=1}^{7}\binom{7}{k} \cdot 2^{7-k} = 2,059$ entries is built.

V. Knowledge-based Paranoia Search

We represent the knowledge in the players as sets. To introduce the reasoning on the sets we give a brief example. Suppose we have the following deal

$P_0: \heartsuit J, \diamondsuit J, \heartsuit A, \heartsuit K, \heartsuit 9, \heartsuit 7, \clubsuit A, \clubsuit 8, \clubsuit 7, \spadesuit A,$
$P_1: \clubsuit J, \spadesuit J, \heartsuit Q, \clubsuit T, \clubsuit K, \clubsuit Q, \spadesuit T, \spadesuit 7, \diamondsuit Q, \diamondsuit 7,$
$P_2: \heartsuit T, \heartsuit 8, \clubsuit 9, \spadesuit K, \spadesuit Q, \spadesuit 9, \spadesuit 8, \diamondsuit A, \diamondsuit T, \diamondsuit 8,$
Skat: $\diamondsuit K, \diamondsuit 9$

with the opponent P_2 to issue the first card. The game that is being played is $\heartsuit$.

We have the following initial knowledge for P_2: $h_0 = h_1 = \{\}$, $h_2 = \{\heartsuit T, \heartsuit 8, \clubsuit 9, \spadesuit K, \spadesuit Q, \spadesuit 9, \spadesuit 8, \diamondsuit A, \diamondsuit T, \diamondsuit 8\}$
$pool = \{\clubsuit J, \spadesuit J, \heartsuit J, \diamondsuit J, \heartsuit A, \heartsuit K, \heartsuit Q, \heartsuit 9, \heartsuit 7, \clubsuit A, \clubsuit T, \clubsuit K, \clubsuit Q, \clubsuit 8, \clubsuit 7, \spadesuit A, \spadesuit T, \spadesuit 7, \diamondsuit K, \diamondsuit Q, \diamondsuit 9, \diamondsuit 7\}$
$skat = \{\}$, $declarerorskat = \{\}$, $partnerorskat = \{\}$
$noskat = \{\clubsuit J, \spadesuit J, \heartsuit J, \diamondsuit J, \heartsuit A, \heartsuit T, \heartsuit K, \heartsuit Q, \heartsuit 9, \heartsuit 8, \heartsuit 7, \clubsuit A, \spadesuit A, \diamondsuit A\}$

The declarer sees table card $\diamondsuit A$ and updates his knowledge sets to $h_0 = \{\heartsuit J, \diamondsuit J, \heartsuit A, \heartsuit K, \heartsuit 9, \heartsuit 7, \clubsuit A, \clubsuit 8, \clubsuit 7, \spadesuit A\}$, $h_1 = h_2 = \{\}$, $skat = \{\diamondsuit K, \diamondsuit 9\}$
$pool = \{\clubsuit J, \spadesuit J, \heartsuit Q, \heartsuit T, \heartsuit 8, \clubsuit T, \clubsuit K, \clubsuit Q, \clubsuit 9, \spadesuit T, \spadesuit K, \spadesuit Q, \spadesuit 9, \spadesuit 8, \spadesuit 7, \diamondsuit T, \diamondsuit Q, \diamondsuit 8, \diamondsuit 7\}$

The opponent player 1 now encounters $\diamondsuit A, \heartsuit A$ on the table and updates his knowledge sets to
$h_0 = \{\}$, $h_1 = \{\clubsuit J, \spadesuit J, \heartsuit Q, \clubsuit T, \clubsuit K, \clubsuit Q, \spadesuit T, \spadesuit 7, \diamondsuit Q, \diamondsuit 7\}$, $h_2 = \{\}$, $skat = \{\}$
$pool = \{\heartsuit J, \diamondsuit J, \heartsuit K, \heartsuit T, \heartsuit 9, \heartsuit 8, \heartsuit 7, \clubsuit A, \clubsuit 9, \clubsuit 8, \clubsuit 7, \spadesuit A, \spadesuit K, \spadesuit Q, \spadesuit 9, \spadesuit 8\}$
$declarerorskat = \{\}$, $partnerorskat = \{\diamondsuit T, \diamondsuit K, \diamondsuit 9, \diamondsuit 8\}$
$noskat = \{\clubsuit J, \spadesuit J, \heartsuit J, \diamondsuit J, \heartsuit T, \heartsuit K, \heartsuit Q, \heartsuit 9, \heartsuit 8, \heartsuit 7, \clubsuit A, \spadesuit A\}$

The general approach to solve a card game with randomness in the deal and partial information is to compute approximate Nash equilibria e.g., using counterfactual regret minimization [5]. As this computation appears not to be feasible within the given time limits to play a card, approximations have to be found - of which inference, sampling, paranoid search, etc. are some examples. We identified two different approaches for the search of playing cards with uncertainty. One is to generate a set of possible (or all) worlds coherent with the generated knowledge, and, then, to merge the result, possibly improved with dominance checks [7]. This is what is done during endgame play [13]. When the set of worlds is statistically Monte-Carlo sampled wrt. the knowledge of the distribution bias can be given to the distribution. However, the approach often misses the best playing card in early stages of the game, when less knowledge is available. The number of declarer cards unknown to the opponents is important. Is it only one card, the is no need to cut low, as the card will likely be put into the Skat. If the opponents issues, he might use a *sharp 10* to win the game. One can also determine if a victory can be enforced or Schneider avoided.

Furtak first describes paranoid search for creating hand databases [10]. Edelkamp [13] also presented a first attempt for conducting a search for a forced win against all odds, aimed at the first card of the Skat game. While interesting for solving Skat puzzles in newspapers, the running time for the analysis, however, was way too large to assist actual play given the restrictions to select a card imposed by the play clock.

To alleviate the computational burden, we propose the search to be initiated only after a few tricks have been played. The algorithm has been adapted to the knowledge already inferred by the Skat AI. It, thus, takes as an input *knowledge sets* [13] corresponding to the inference that player P_i must have cards C_j (not) in his hand. This knowledge is inferred e.g., by unrealistically bad skats, players not obeying trump or non-trump cards and by playing conventions (putting the lowest-valued card in the declarer's trick, and the highest-value trick to the one of my partner, with some exceptions). As with many other parts of the Skat AI, for efficiency reasons, sets of cards are encoded as bit-vectors of length 32 (unsigned int). This allows fast bit manipulation, such as card selection and copying. The minimax alpha-beta simulating *moving test-driver* search algorithm [8] to analyze partial information trump games is implemented as a binary search (see Figure 2) over an AND/OR tree decision procedure (see Figure 3) that returns, whether or not the declarer can win the game according to a given contract limit. It progresses belief sets for partial information. The knowledge-based Paranoia search (KBPS) algorithms are applied in forehand, in middlehand, and in rearhand positions of the players. Besides updating knowledge vectors, scoring values, current contract limit, the call has to respect played cards on the table to trigger a correct analysis. By monitoring server logs online during play, we validated the working of the algorithm: once a win has been found it persists to the end, in many cases long before the human opponent recognized that he is lost.

a) Paranoia Search for the Declarer: The KBPS worst-case analysis for the declarer is used in trump games. Its implementation is a loop over backtracking moving test driver branch-and-bound procedure to find the optimal game value. As we use the search option dynamically, the algorithm is initiated after a fixed number k of played cards. In the overall architecture it acts as a prioritized killer card proposal that warrants a forced win. Paranoia search takes the partially played game, and a set of possible worlds as a parameter, encoded as knowledge sets, and contract bound. We limit the uncertain knowledge to the sets of *free* cards that are still to be distributed among the two opponent players. All *fixed* cards are assigned to one hand. Fig. 4 shows the implementation of the declarers' KBPS backtracking algorithm at an OR node for the first opponent in the AND-OR partial observable search tree. It determines if a game can be won against all worlds according to a given score bound $limit$ as fixed by the overall binary search. For the sake of simplicity, we omit code for transposition table pruning [32] and for pruning of equivalent cards [6]. As the declarer knows that Skat (except for *Hand* games), as with the above overall knowledge representation and reasoning example there are 3 knowledge sets provided to the player: $pool$, denoting all remaining cards not yet known on which opponent hand they reside, h_1, cards already known to be in the 1st opponent hand, h_2, cards already known to be in the 2nd opponent hand. Furthermore, we have $avail$: hand cards playable according to the rules of Skat, obeying trump and suit; $index$, bit: selected card, for being played; $played$: cards already played; w: winner of trick; $i_0, \ldots, i_2$: table cards by players; r_1, r_2 number remaining cards available; $limit$: current bound for game value; $score$: card value of table cards; $aspts$: point total for the declarer (according to the given knowledge of the Skat); $gspts$; point total for the opponents.

```
solve(hand0,hand1,hand2,doublehand,played,as,gs,table)
  left = as-1, right =  120
  while (true)
    if (left == right-1) return right
    limit = (left+right)/2
    x = run(hand0,hand1,hand2,doublehand,played,as,gs,table)
    if (!x) right = limit else left = limit
```

Fig. 2. Moving test driver for declarer knowledge-based paranoia search; hand0 are the cards of the declarer, hand1/hands2 opponent cards known to him, doublehand is the pool; unknown, which hands the cards belong.

```
run(hand0,hand1,hand2,doublehand,p,as,gs,table)
  h[0] = hand0; h[1] = hand[1]; h[2] = hand2, i = table;
  aspts = as; gspts = gs, pool = doublehand; played = p
  return AND(hand0);
```

Fig. 3. Running declarer knowledge-based paranoia search; starting and or tree search after set some global backtracking variables.

The KBPS algorithm searches the tree of playable cards, and branches wrt. the set of known cards and the current belief, while respecting the rules of play and the number of cards that a player can have. If suits are not obeyed, knowledge vectors for cards available to each hand are updated during the search. Before cards are selected from the pool of cards available to both players they are assigned to one opponents' hand.

The algorithm can be extended to cover more knowledge inference options like playing conventions for the opponents such as giving the highest-valued card to a trick that goes to the partner, and a lower-valued card to a trick the declarer.

If the capacity of a hand is exceeded, we encounter a dead-end and a backtrack is initiated. In other words, if more cards are assigned to the player than his hand can hold, the entire subtree is pruned. By the virtue of enumeration of all card combinations, the algorithm computes the game-theoretical partial information (minimax) score, assuming optimal play of the players. The transposition table and equivalent card pruning are implemented in a way not to violate this outcome.

A proof that a win is forced and will not be lost during subsequent play can be done by induction on the number of remaining cards to be played, but is quite obvious, as the game-theoretical minimax value is computed at the root node.

```
OR1(avail)
  avail = playable(avail,h[1],i)
  while (avail)
    index = select(avail);
    bit = (1<<index);
    h2 = h[2]; o = pool
    if (c = first-card-on-table(i))
      if (trump & (1 << c))
        if (|trump & bit| == 0)
          h[2] |= trump & pool;
          if (exceeded(h[2]))
            h[2] = h2;
            avail &= ~bit;
            continue;
          pool &= ~h[2];
     else
       if (|suit(c) & bit| == 0)
          h[2] |= suit(c) & pool;
          if (exceeded(h[2]))
            h[2] = h2;
            avail &= ~bit;
            continue;
          pool &= ~h[2];
    if (exceeded(h[1]|bit))
      (h[2],pool) = (h2,f);
      avail &= ~bit;
      continue;
    (h1,p,i[1],r) = (h[1],played,index,-1);
    pool &= ~bit; h[1] &= ~bit; played |= bit;
    if (endoftrick(i))
      w = winner(2,0,1);
      score = value(i);
      gspts += w ? score ; 0;
      aspts += w ? 0 : score;
      (i0,i2) = (i[0],i[2]); i[0] = i[1] = i[2] = -1;
      r = (gspts >= 120-limit) ? 0 :
          (aspts > limit) ? 1 :
          (w == 0) ? AND(h[0]) :
          (w == 1) ? OR1((pool|h[1]) & ~h[2]) :
         OR2((pool|h[2]) & ~h[1]);
      i[0] = i0; i[2] = i2;
      gspts -= w ? score : 0;
      aspts -= w ? 0 : score;
      (i[1],h[1],h[2],pool,played) = (-1,h1,h2,o,p);
    avail &= ~bit;
    if (r == 0) return 0;
  return 1;
```

Fig. 4. Declarer KBPS at an OR search node for the first opponent's selection of a card; other search node implementations are simpler or similar.

As long as the knowledge is exact, i.e., given that no false information is contained in the knowledge vectors, then the algorithm progressing the vector does not falsify it. We are not claiming correctness, if no forced win is found, then the game continues with other card recommendations.

Proposition 1 (Soundness KBPS for Declarer Play): Given that the knowledge provided in the knowledge vectors is valid at invocation time of the algorithm, once value 1 is returned by the KBPS declarer algorithm (cf. Figure 4), the game is won by the declarer and this forced win will manifest during subsequent trick-taking play. If the algorithm optimizes the number of points in the moving test driver, the declarer will receive more than points the computed limit.

b) Paranoia Search for Schneider & Schwarz: When a game can be won to the contract of 61 points, it is desirable to aim at Schneider (90 pts) or Schwarz (120 pts). This is done by restarting the analysis with a higher contract, once the one for the current limit has been proven to be a win.

c) Approximate Paranoia Search: The worst-case analysis has two major limitations. As stated in Theorem 1, the AIs act *in paranoia*. Suppose that all non-trump card of a suit are neither in the declarers hand nor in the Skat, then even extreme distributions of the cards with all cards on either hand have to be accounted for in the analysis. The probability for this case, however, is only 1.5625%. The virtue of good Skat play is to play well against most likely and not all card distributions. For the approximate KBPS algorithm we, therefore, demand that certain distributions of cards are unlikely, and should be excluded from the search. Secondly, the running time is larger in case of more uncertainty, so that belief space measured in the number of worlds the AI plays against, may hinder finishing a complete KBPS exploration in time.

Both objections can be met together by limiting the cards that can be assigned to each hand. This is the basis of *approximate knowledge-based paranoia search* (AKBPS), that poses constraints on the cards distributions allowed on each hand, or —for implementation purposes— enforces some cards assigned to a hand. Of course, the theorem no longer holds, as there are some worlds that are not considered, still the observation is, that early suggestions of cards that wins against all but extreme worlds are extremely valuable. In contrast, Furtak used lower bounds, and set the declarer cut-off to 57.

d) Paranoia Search for the Opponents: Extending the approach from the declarers' point of view to the ones of the opponents is tricky, mainly due to the presence of the unknown Skat. For example, if one of the other players does not obey, it is no longer immediate that the card is on the remaining player's hand, as it can reside in the Skat. In the knowledge-based paranoia search algorithm, illustrated for the case of the declarer's AND node in Figure 5 (for efficiency reasons, we are using many bitvector set operations!), this leads to the introduction of further knowledge vectors. We now have five sets that are updated denoting that the declarer or the Skat has a card, or that the opponent, or the Skat has a card of the pool of remaining cards, only if taken or a card is definitely known to be on the hand, e.g., by selecting it, it is moved. In some respects, the knowledge sets ($declarerorskat$ and $partnerorskat$) are caches for the main pool of cards (*pool*) for the remaining players. In some of the conditions applied we take care that no more cards are moved to a hand than it can cope with.

If one opponent sees a definite win, this does not mean that the other opponent sees it as well. Given a different set of hand cards he may have very different knowledge on the distribution of cards. As it is defined, it requires one defender to assume that his partner will intentionally play poorly. Again, soundness can be proven by induction of the remaining cards to be played, and the observation that a search tree with less remaining cards is part of a search tree with more remaining cards, leading to a forced win. According to the uncertainty in the Skat there are three pools of cards that reflect the rising knowledge instead of one.

Proposition 2 (Soundness KBPS for Opponent Play): Given that the knowledge provided in the knowledge vectors is valid at invocation time of the algorithm, once a card is returned by the KBPS opponen algorithm (cf. Figure 5), the game is won by the opponent and this forced win will manifest during subsequent trick-taking play. If the algorithm optimizes the number of points in the moving test driver, the declarer will not receive more points than the computed limit.

e) Worst-Case Analysis for Avoiding Schneider/Schwarz: In opponent play, using a paranoid assumption on the card play is less effective than for the declarer play, and often applies to the endgame analysis. When the game is won by the declarer, however, KBPS, frequently applies to avoid a high loss with 90 declarer points, called Schneider, or a maximum loss with 120 declarer points. Therefore, once the contract of the declarer has been achieved, we use KBPS in opponent play with a scoring limit for Schneider/Schwarz.

VI. Experiments

The Skat AI is written in C++, compiled with `gcc` version 4.9.2 (optim. level `-O2`). Each player client runs on 1 core of an Intel Xeon Gold 6140 CPU @ 2.30GHz.

We determine the average of the game value according to the extended Seeger-(Fabian-)System, the internationally agreed DSKV standard for evaluating game play, normalized to a series of 36 games. The score is based on the number of wins and losses of each player in the series, and the game value of the games being played. For a single game g, the outcome is $V(D,g)$, if the game is won for the declarer D and $-2 \cdot V(D,g)$, if it is lost. In a series of games $G = g_1, \ldots, g_X$ these values are added for each player, so that $V(A,G) = V(A,g_1) + \ldots + V(A,g_X)$. The evaluation strength of Player A wrt. B and C is $V(A,G) + 50 \cdot (\#win(A,G) - \#loss(A,G)) + 40 \cdot (\#loss(B,G) + \#loss(C,G))$.

a) Database Play: The obtained results on 50,000 human expert trump games are presented in Tables I– Table III. For the three valid combinations of AI/Human bidding/discarding/game announcement we separate between the play with and without the support of Paranoia search. In the columns we further partition the game outcomes with respect to the declarer in the $i)$ original Human game play, $ii)$ an open card solver (that we call Glassbox), and $iii)$ AI trick-taking selfplay.

In Table I we see that with the support of the Paranoia search, the declarer is able to win $42,228 - 41,765 = 463$ more than the AIs without Paranoia search and far more than the Humans in their play $42,228 - 41,283 = 945$. This is a significant progress, given that the number of wins was already high. The number of games won (and the extended Seeger values) were higher than the ones obtained by the humans.

The AIs with KBPS show better winning ratios than the humans, and a significant positive effect on the playing strength in extended Seeger score: for AI bidding and Skat putting almost $1,000$ points. With up to 50% additional time, there is a computational trade-off, but in server play selecting the card to play remains below 5s. In contrast to Null games, automated Skat putting in trump is worse to the Human one, and, therefore, subject to further research. For AI bidding the total of wins/losses is not matching the total number of games, as some games might be folded.

We analyzed another set of over 75 thousand human expert games (different server to ours, all kinds of games). We varied the card number k to start approximate KBPS at card k and KBPS at card $k+3$. At $k=6$ we reached 1000.24 extended Seeger scoring points. At $k=3$ we could slightly improve the value to 1001.39, in a tradeoff of a slowdown factor 2-3.

	Human Wins	Glassbox Wins	AI Wins	+Paranoia Opponents	−Paranoia Opponents
−Paranoia Declarer	false	false	false	2,563	2,530
	false	false	true	2,208	2,241
	false	true	false	231	226
	false	true	true	2,658	2,663
	true	false	false	3,438	3,407
	true	false	true	5,540	5,571
	true	true	false	975	970
	true	true	true	31,285	31,290
Total +PO	41,283	35,149	41,691	48,898	-
Total -PO	41,283	35,149	41,765	-	48,898
Total Score				977.76	980.07
Total Time				37h:51m	31h:01m
+Paranoia Declarer	false	false	false	2,460	2,458
	false	false	true	2,315	2,313
	false	true	false	193	194
	false	true	true	2,696	2,695
	true	false	false	3,236	3,240
	true	false	true	5,742	5,738
	true	true	false	785	788
	true	true	true	31,475	31,472
Total +PO	41,283	35,149	42,228	48,898	-
Total -PO	41,283	35,149	42,218	-	48,898
Total Score				990.25	991.79
Total Time				43h:32m	42h:06m

TABLE I
SKAT AI REPLAYING 50,000 HUMAN TRUMP GAMES WITH AND WITHOUT KBPS, USING AI BIDDING GAME SELECTION AND SKAT PUTTING. SCORE IS EXTENDED SEEGER, AVERAGED OVER 36 GAMES. TABLE SPLIT BASED ON KBPS BEING APPLIED FOR DECLARER AND OPPONENTS. TOTAL OF GAMES IS SMALLER BECAUSE OF $1,102$ FOLDINGS (NO BID).

	Human Wins	Glassbox Wins	AI Wins	+Paranoia Opponents	−Paranoia Opponents
−Paranoia Declarer	false	false	false	3,854	3,822
	false	false	true	2,779	2,811
	false	true	false	198	195
	false	true	true	1,040	1,043
	true	false	false	2,137	2,110
	true	false	true	5,271	5,298
	true	true	false	965	953
	true	true	true	33,756	33,768
Total+PO	42,129	35,959	42,846	50,000	-
Total−PO	42,129	35,959	42,920	-	50,000
Total Score				953.31	955.91
Total Time				34h:20m	25h:31m
+Paranoia Declarer	false	false	false	3,784	3755
	false	false	true	2,849	2,878
	false	true	false	184	128
	false	true	true	1,054	1,056
	true	false	false	1,998	1,973
	true	false	true	5,410	5,435
	true	true	false	761	752
	true	true	true	33,960	33,969
Total + PO	42,129	35,959	43,273	50,000	-
Total - PO	42,129	35,959	43,338	-	50,000
Total Score				963.35	965.53
Total Time				47h:21m	38h:05m

TABLE II
SKAT AI REPLAYING 50,000 HUMAN TRUMP GAMES WITH AND WITHOUT KBPS USING HUMAN BIDDING, GAME SELECTION, AND SKAT PUTTING. SCORE IS EXTENDED SEEGER, NORMALIZED TO 36 GAMES. TABLE SPLIT KBPS BEING APPLIED FOR DECLARER / OPPONENTS.

VII. Conclusion

We have seen an improvement for knowledge inference in searching partial information games. The novelty is to include knowledge representation and reasoning into the backtrack partial-information game-tree search. In contrast to *perfect-information Monte-Carlo sampling* used by many AI card playing systems [6], with a search for a sampled set of worlds, the KBPS search algorithm operates against all possible worlds in one search tree, avoiding the fusion of different card

	Human Wins	Glassbox Wins	AI Wins	+Paranoia Opponents (PO)	-Paraonoia Opponents (PO)
−Paranoia Declarer	false	false	false	3,941	3,894
	false	false	true	2,519	2,566
	false	true	false	200	196
	false	true	true	1,211	1,215
	true	false	false	2,611	2,581
	true	false	true	5,502	5,532
	true	true	false	934	923
	true	true	true	33,082	33,093
Total +PO	42,129	35,959	42,314	50,000	-
Total -PO	42,129	35,959	42,406	-	50,000
Total Score				935.23	938.41
Total Time				34h:10m	25h:15m
+Paranoia Declarer	false	false	false	3,853	3,808
	false	false	true	2,607	2,652
	false	true	false	187	185
	false	true	true	1,224	1,227
	true	false	false	2,452	2,415
	true	false	true	5,661	5,698
	true	true	false	732	724
	true	true	true	33,284	33,292
Total +PO	42,129	35,959	42,776	50,000	-
Total -PO	42,129	35,959	42,869	-	50,000
Total Score				947.41	950.39
Total Time				37h:39m	46h:49m

TABLE III
SKAT AI REPLAYING 50,000 HUMAN TRUMP GAMES WITH AND WITHOUT KBPS USING HUMAN BIDDING, AND AI SKAT PUTTING. SCORE IS EXTENDED SEEGER NORMALIZED TO 36 GAMES. TABLE SPLIT ON KBPS BEING APPLIED FOR DECLARER / OPPONENTS.

suggestion and resulting in a single card recommendation. It progresses the knowledge in the search tree in an efficient manner, resulting in an optimal search algorithm that is fast enough to be applied in early stages of the game even after a few cards have been played and especially for declarer play, leads to card suggestions that even experienced humans often do not see. If the analysis succeeds, this *killer card* is forced. If not, other card recommendations like expert rules or end game play apply. Although exemplified for *Skat*, the contribution is general to work for other multi-player card games like *Spades*, *Hearts*, *Tarot*, *Marias*, *Ulti*, or *Bridge*, and likely to other domains. The results in increased playing strength especially for the declarer are unexpectedly promising. Even experienced top players were often surprised to lose against the automated play to victory, once a forced win had been found. More research is needed to increase the knowledge in the search tree, especially for opponent play.

Future work is to extend the reasoning in Paranoia search further. To improve the reasoning about the knowledge that is forwarded in the search. We saw that paranoia search is a worst-case analysis. It is not difficult to see that a best-case analysis also applies, e.g., to determine if —in the view of an opponent— the declarer can win by certain. In this case they can switch to save Schneider. There are further research avenues towards approximate KBPS search, as the virtue of good card play is a card selection to win not against all but most possible distributions. Approximate paranoia search may be extended to work hierarchically, looking for say forced wins against all but a rising number of less extreme cases for the distribution of the remaining cards. With further algorithmic refinement, we envision using (approximate) KBPS to be able to decide, if a *Grand* is safe and should be played. The only matter is to improve efficiency, either by algorithm engineering, exploitation of parallel hardware, or by neglecting lost cards from the analysis.

```
AND(avail)
  while (avail)
    index = select(avail);
    bit = (1<<index);
    (h0,h2,o,as,ms) =
         (h[0],h[2],pool,declarerorskat,partnerorskat);
    if (c = first-card-on-table(i))
      if (trump & (1 << c))
        if (|trump & bit| == 0)
          partnerorskat |= trump & pool;
          if (exceeded(partnerorskat))
            h[2] = h2; partnerorskat = ms;
            avail &= ~bit;
            continue;
          pool &= ~partnerorskat;
          h[2] |= noskat & partnerorskat;
          if (exceeded(h[2]))
            (h[2],pool,partnerorskat) = (h2,o,ms);
            avail &= ~bit;
            continue;
          partnerorskat &= ~h[2];
      else
        if (|suit(c) & bit| == 0)
          partnerorskatat |= suit(c) & o;
          if (exceeded(partnerorskat))
            (h[2],t,partnerorskat) = (h2,o,ms);
            avail &= ~bit;
            continue;
          pool &= ~partnerorskat;
          h[2] |= noskat & partnerorskat;
          if (exceeded(h[2]))
            (h[2],pool,partnerorskat) = (h2,o,ms);
            avail &= ~bit;
            continue;
          partnerorskat &= ~h[2];
    if (exceeded(h[0]|bit))
     (h[2],partnerorskat,pool) = (h2,ms,o);
     avail &= ~bit;
     continue;
    p = played; pool &= ~bit;
    h[0] &= ~bit;h[2] &= ~bit;
    declarerorskat &= ~bit;
    partnerorskat &= ~bit;
    played |= bit; i[0] = index; r = -1;
    if (endoftrick)
      w = winner(1,2,0);
      score = value(i);
      ap = aspts; gp = gspts;
      gspts += w ? score: 0;
      aspts += !w ? score: 0;
      t1 = i[1], t2 = i[2];
      if (|played| == 30) aspts += value(~played);
      i[0] = i[1] = i[2] = -1;
      r = (gspts >= 120-limit) ? 0:
          (aspts > limit) ? 1:
          (w==0)? AND((pool|declarerorskat|h[0]) & ~h[2]):
          (w==1)? OR1(h[1]):
         OR2((o|partnerorskat|h[2]) & ~h[0]);
      i[1] = t1; i[2] = t2;
      gspts = gp; aspts = ap;
    else r = OR1(feasible(h[1],i));
    (h[0],h[2],played,declarerorskat,partnerorskat,pool) =
       (h0,h2,p,as,ms,o);
    avail &= ~bit; i[0] = -1;
    if (r == 1) return 1;
  return 0;
```

Fig. 5. Opponent KBPS at an AND search node for a declarer's selection of a card; other search node implementations are simpler or similar.

Acknowledgements: Thanks to Rainer Gößl, a world-class Skat player, who helped with his Skat expertise. This work has been supported by the OP RDE funded project Research Center for Informatics No.: CZ.02.1.01/0.0./0.0./$16_0 19/0000765$

REFERENCES

[1] J. Schaeffer, "The games computers (and people) play," in *Fortieth Anniversary Volume: Advancing into the 21st Century*, ser. Advances in Computers, M. V. Zelkowitz, Ed. Elsevier, 2000, vol. 52, pp. 189–266. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0065245800800194

[2] D. Silver and A. H. et al., "Mastering the game of Go with deep neural networks and tree search," *Nature*, vol. 529, p. 484, 2016.

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

[4] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. Lillicrap, K. Simonyan, and D. Hassabis, "Mastering Chess and Shogi by self-play with a general reinforcement learning algorithm," arxiv, Tech. Rep. 1712.018, 2017.

[5] M. Bowling, N. Burch, M. Johanson, and O. Tammelin, "Heads-up limit hold'em poker is solved," *Commun. ACM*, vol. 60, no. 11, pp. 81–88, 2017. [Online]. Available: http://doi.acm.org/10.1145/3131284

[6] M. Ginsberg, "Step toward an expert-level Bridge-playing program," in *IJCAI*, 1999, pp. 584–589.

[7] T. Cazenave and V. Ventos, "The $\alpha\mu$ search algorithm for the game of bridge," *CoRR*, vol. abs/1911.07960, 2019. [Online]. Available: http://arxiv.org/abs/1911.07960

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

[9] C. Solinas, D. Rebstock, and M. Buro, "Improving search with supervised learning in trick-based card games," in *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019*. AAAI Press, 2019, pp. 1158–1165. [Online]. Available: https://doi.org/10.1609/aaai.v33i01.33011158

[10] T. M. Furtak, "Symmetries and search in trick-taking card games," Ph.D. dissertation, University of Alberta, 2013.

[11] S. Edelkamp, "Challenging human supremacy in Skat," in *Proceedings of the Twelfth International Symposium on Combinatorial Search, SOCS 2019, Napa, California, 16-17 July 2019*, 2019, pp. 52–60. [Online]. Available: https://aaai.org/ocs/index.php/SOCS/SOCS19/paper/view/18328

[12] ——, "Dynamic play via suit factorization search in Skat," in *KI*, ser. Lecture Notes in Computer Science, vol. 12325. Springer, 2020, pp. 18–32. [Online]. Available: https://doi.org/10.1007/978-3-030-58285-2_2

[13] ——, "Representing and reducing uncertainty for enumerating the belief space to improve endgame play in skat," in *ECAI*, 2020, pp. 395–402. [Online]. Available: https://doi.org/10.3233/FAIA200118

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

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

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

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

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

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

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

[21] S. Kupferschmid and M. Helmert, "A Skat player based on Monte-Carlo simulation," in *Computers and Games*, 2006, pp. 135–147.

[22] M. Buro, J. R. Long, T. Furtak, and N. 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*, 2009, pp. 1407–1413. [Online]. Available: http://ijcai.org/Proceedings/09/Papers/236.pdf

[23] T. Keller and S. Kupferschmid, "Automatic bidding for the game of Skat," in *KI*, 2008, pp. 95–102.

[24] C. Solinas, D. Rebstock, and M. Buro, "Improving search with supervised learning in trick-based card games," *CoRR*, vol. abs/1903.09604, 2019. [Online]. Available: http://arxiv.org/abs/1903.09604

[25] D. Rebstock, C. Solinas, and M. Buro, "Learning policies from human data for Skat," *CoRR*, vol. abs/1905.10907, 2019. [Online]. Available: http://arxiv.org/abs/1905.10907

[26] D. Rebstock, C. Solinas, M. Buro, and N. R. Sturtevant, "Policy based inference in trick-taking card games," *CoRR*, vol. abs/1905.10911, 2019. [Online]. Available: http://arxiv.org/abs/1905.10911

[27] N. R. Sturtevant and A. M. White, "Feature construction for reinforcement learning in hearts," in *Computers and Games*. Springer, 2006, pp. 122–134. [Online]. Available: https://doi.org/10.1007/978-3-540-75538-8_11

[28] G. Cohensius, R. Meir, N. Oved, and R. Stern, "Bidding in spades," in *ECAI*, 2020, pp. 387–394. [Online]. Available: https://doi.org/10.3233/FAIA200117

[29] E. R. Berlekamp, J. H. Conway, and R. K. Guy, *Winning Ways for Your Mathematical Plays, Vol. 1–4*. A K Peters, 2001.

[30] N. R. Sturtevant and R. E. Korf, "On pruning techniques for multi-player games," in *AAAI/IJCAI*, 2000, pp. 201–207.

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

[32] A. Reinefeld and T. A. Marsland, "Enhanced iterative-deepening search," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 16, no. 7, pp. 701–710, 1994. [Online]. Available: https://doi.org/10.1109/34.297950