Automatic Bidding for the Game of Skat *

Thomas Keller and Sebastian Kupferschmid

University of Freiburg, Germany
{tkeller, kupfersc}@informatik.uni-freiburg.de

Abstract. In recent years, researchers started to study the game of Skat. The strength of existing Skat playing programs is definitely the card play phase. The bidding phase, however, was treated quite poorly so far. This is a severe drawback since bidding abilities influence the overall playing performance drastically. In this paper we present a powerful bidding engine which is based on a k-nearest neighbor algorithm.

1 Introduction

Although mostly unknown in the English-speaking world, the game of Skat is the most popular card game in continental Europe, surpassed in world-wide popularity only by Bridge and Poker. With about 30 million casual players and about 40,000 people playing at a competitive level, Skat is mostly a German phenomenon, although national associations exist in twenty countries on all six inhabited continents. It is widely considered the most interesting card game for three players.

Despite its popularity, only in recent years researchers started to study the game of Skat. This is not due to lack of challenge, as Skat is definitely a game of skill – significant experience is required to reach tournament playing strength. Like Bridge, Skat consists of two playing phases: after the bidding, which determines the alliance and the trump for the game, the actual card playing begins.

The best performing skat playing programs we are aware of are the implementations of Kupferschmid and Helmert [1] and Schäfer [2]. The strength of these programs is the card playing phase. Both programs also have a bidding module, but in both cases its performance is rather poor. The problem with poor bidding abilities is that it influences the overall performance of a Skat player dramatically. The reason for this is that a victory can easily be turned into a loss if the wrong trump was chosen. In this paper, we investigate how a k-nearest neighbor algorithm can be used to build a more sophisticated bidding engine for the game of Skat.

The paper is structured as follows. Section 2 briefly introduces the rules of Skat. Section 3 focuses on the bidding phase and gives a general architecture for a bidding engine. Afterward a k-nearest neighbor based algorithm for the bidding is presented and applied to discarding in Section 5. Further improvements are described afterwards. Section 7 concludes.

* This work was partly supported by the German Research Council (DFG) as part of the Transregional Collaborative Research Center "Automatic Verification and Analysis of Complex Systems" (SFB/TR 14 AVACS). See http://www.avacs.org/ for more information.

2 Skat

In this section we provide a brief overview of the skat rules. For a more detailed description, we refer to the official rules [3].

Skat is a three-player imperfect information game played with 32 cards, a subset of the usual Bridge deck. At the beginning of a game, each player is dealt ten cards, which must not be shown or communicated to the other players. The remaining two cards, called the *skat*, are placed face down on the table. Like in Bridge, each hand is played in two stages, *bidding* and *card play*.

The bidding stage determines the alliance for this hand and proceeds as follows: two players announce and accept increasing bids until one of them passes. The winner of the first bidding phase now competes in the same way with the third player. The successful bidder of the second bidding phase, henceforth called the *declarer*, plays against the other two. In this process, the maximum bid a player can announce depends on the kind of game (grand, ♣, ♠, ♡, ♢, null) the player wants to play and a factor determined by the jacks the player holds. To avoid confusion about "game" and "kind of game", we henceforth call the latter "trump". The declarer decides on the trump. Before declaring the game, the declarer may pick up the skat and then discard any two cards from his hand, face down. These cards count towards the declarer's score.

Card play proceeds as in Bridge, except that the trumps and card ranks are different. In grand games, the four jacks are the only trumps. In suit games, the four jacks and the seven other 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 the declarer must score more points than the opponents (i. e. at least 61 points) to win. Null games are an exception and follow *misère* rules: the declarer wins iff he scores no trick. Note that we do not deal with null games in this paper since they are rather trivial, from a bidder's perspective, because there are efficient rule-based criteria to decide whether a null game can be won or not.

3 Bidding and Discarding

From a player's perspective, the problem of bidding is to decide if there is a trump to win the game. This decision is only based on the cards the player holds at the beginning of the game, as this is the only information a player has available. To estimate a game's outcome as precisely as possible is crucial for a good overall playing performance, but very difficult under the given information as well. Some factors that determine a hand's strength are rather obvious, such as the number of trumps, but in fact there are more subtle intricacies that influence the potential of a player's hand enormously.

Discarding cards is a similar problem. If the declarer picks up the skat, he has to decide which two of his twelve cards to discard. Again some decisions seem rather obvious, e. g., a player should discard a ten when he does not hold the ace of the same suit, but in general the choice is far from simple.

The bidding engine of Kupferschmid's and Helmert's Double Dummy Skat Solver (DDSS) [1] is based on a least mean square error algorithm (cf. [4]), which tries to separate the winnable from the non-winnable games by learning a linear function. The

problem with this approach is that it is very unlikely that Skat games can be linearly separated. This is the main reason for DDSS rather poor bidding behavior.

The bidding system we are presenting in this paper is based on a k-nearest neighbor algorithm (KNN). The advantage of this machine learning algorithm is that it is well suited to approximate difficult separation functions. Our bidding module consists of several different *evaluation functions* $V^t : S \to \{0, \ldots, 120\}$, one for each $t \in T = \{\text{grand}, \clubsuit, \spadesuit, \heartsuit, \diamondsuit\}$. Such a function maps each set of ten cards (a player's hand) $s \in S$ to a number between 0 and 120 (the possible outcome of the game), estimating the potential of the given cards if trump t is played. With these evaluation functions it is possible to describe our bidding system as $C(s) = \arg\max_{t \in T}\{V^t(s) \mid V^t(s) > th\}$. If none of the estimates $V^t(s)$ is above the threshold th (usually $th = 60$) then $C(s)$ evaluates to $\emptyset$. In case that there are more than one t, the t allowing the highest bid is selected.

With our bidding system it is also possible to solve the discarding problem. As said before, the problem here is to select two cards out of twelve so that the remaining ten cards are as strong as possible. For every of the 66 possible subsets that contain ten cards our bidding system is applied. The cards that yield the highest estimation of the outcome are discarded. A more detailed description, also including some refinements, of this procedure is presented in Sec. 5.

4 A KNN based Bidding System

The k-nearest neighbor algorithm belongs to the group of instance-based learning algorithms, which have in common that a hand is classified by comparing it to instances for which the correct classification is known. These instances form the *knowledge base*. To be able to compare different hands, a metric on the set of instances has to be defined. Depending on the trump t, we first project an instance s to a numeric feature vector $\langle f_1^t(s), f_2^t(s), \ldots, f_8^t(s)\rangle \in \mathbb{R}^8$. The distance to another instance is then defined as the Euclidean distance of the corresponding normalized feature vectors.

It should be mentioned that, due to the imperfect information, it is not possible to design a bidding system recognizing every winnable game as winnable. Even an optimal bidder will sometimes lose a game. The task of developing adequate features lies in finding features that are significant to describe a hand but still leave the feature space as small as possible. Only jacks and non-jack trumps, the most basic factors for the potential of a hand, are represented twice in the features, both as a simple quantity function and a valuation function as can be seen in Table 1. The other features describe other important aspects of a hand.

To classify a given hand, the distance between that hand and each instance in the knowledge base is computed and the k sets of nearest neighbors are determined. For a trump t the estimated score of a game is the averaged score of all elements of those sets. Note that more than k instances are used for classification, since each set of neighbors contains all instances with the same distance.

Table 1. Features describing a hand s if a game with trump t is played.

feature	description	range
$f_1^t(s)$	number of jacks in s	0–4
$f_2^t(s)$	number of non-jack trumps in s	0–7 (grand: 0)
$f_3^t(s)$	number of non-trump aces and 10s. 10s only if the ace of the same suit is in s	0–6 (grand: 0–8)
$f_4^t(s)$	accumulated points of the cards in s	0–90
$f_5^t(s)$	number of suits not in s	0–3
$f_6^t(s)$	number of non-trump 10s without the ace of the same suit in s	0–3
$f_7^t(s)$	valuation of jacks in s	0–10*
$f_8^t(s)$	valuation of non-jack trumps in s	0–17 (grand: 0)**

* ♣J : 4, ♠J : 3, ♡J : 2, ♢J : 1
** A : 5, 10 : 4, K : 3, Q : 2, 9, 8 and 7 : 1

4.1 Construction of the Knowledge Base

Unlike most other algorithms, KNN does not have a training phase. Instead the computation takes place directly after a query occurs. The only thing needed before using the algorithm is the knowledge base, which is substantial for KNN. In this work, DDSS was used to generate a knowledge base. Therefore we randomly generated 10,000 pairwise distinct games. For each game g and each trump t DDSS was used to approximate the correct game theoretic value. Note that, since Skat is an imperfect information game, it is not possible to efficiently calculate the exact game theoretic value.

To judge the quality of our training and evaluation games it is necessary to know how DDSS works. It is a Monte Carlo based Skat playing program. Its core is a fast state-of-the-art algorithm for perfect information games. In order to decide which card should be played in a certain game position, the following is done. The Monte Carlo approach repeatedly samples the imperfect information (i. e., the cards of the other players). For each of the resulting perfect information games, the game theoretic value is determined. The card which yields the best performance in most of these perfect information games is then played in the imperfect game. Ginsberg [5] first proposed to use a Monte Carlo approach for imperfect information games. However, the problem with this approach is that even if all possible card distributions are evaluated the optimal strategy cannot be computed. This was proven by Basin and Frank [6]. Another problem is that a Monte Carlo player will never play a card in order to gain additional information rather than points. Nevertheless, DDSS has high playing abilities.

For the generation of the knowledge base, the number of samples per card was set to 10. This corresponds to the strength of an average Skat player. The higher the number of samples the better is the playing performance. But since the runtime is exponential in the number of samples, we were forced to use a rather small number of samples.

To evaluate the performance of our bidding system another 5,000 games were randomly generated and calculated by DDSS in the same way. Approximately 60 % of these games were calculated as not winnable. This is a reasonable percentage because we are regarding so called *hand* games[1].

[1] A game is a hand game if the declarer does not pick up the skat

4.2 Choosing the Right Number of Neighbors

In this section we address how to determine an optimal number of neighbors k for this task. Figure 1 shows the average square error depending on the used k. Because of the sweet spot between 8 and 30, we set k to 15. With this k more than 90 % of the games that were calculated as not winnable in the evaluation set were classified correctly. Just 50 % of the winnable games were classified as winnable, but the identification of the correct trump worked in more than 80 % of those cases. Alltogether, 73.2 % of all games were classified in the correct way which is already a pretty good result.

It can be observed that too many games are classified as not winnable. This is especially a problem as all the games are regarded as *hand* games and thus the possible improvement by picking up the skat is not regarded yet. A bidder should rather overestimate a bit and speculate on a slight improvement than bid too carefully. However, the bidding system is already well suited to classify *hand* games.

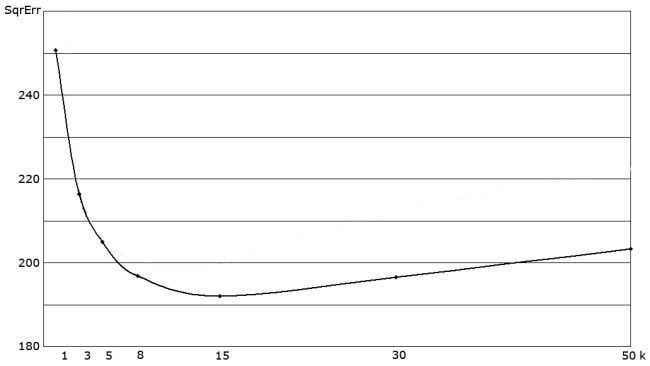

Fig. 1. Average square errors for different k

4.3 Failed Improvement Attempts

In the literature, a couple of improvements to KNN can be found. The most basic improvement is to weigh each nearest neighbor depending on its distance to the query s. The evaluation showed, that the results could not be improved this way. The reason for this is the high degree of incomplete information in Skat. Seemingly, very different hands can describe very similar games due to the cards in the skat or the distribution of the unknown cards. By lowering the influence of further distanced instances on the result, this kind of game is not included strong enough in the result.

Since our knowledge base consists of randomly generated games, there are densely populated areas containing lots of instances with similar feature vectors and sparse areas. A query in a dense area is then classified by more neighbors than a query in a sparse part – even with the same k. Therefore, we created a knowledge base that is uniformly distributed over the feature space – and obtained slightly worse results. The

reason for this is twofold. First, the probability for certain features differs a lot, e. g., it is more likely to hold two jacks than holding four. So, dense areas in our knowledge base are those areas that come up with a higher probability, and most of our evaluation games should fall into dense populated areas as well. Now, most of our evaluation games were classified with less instances and thus a little worse in average, while just a few instances with a rarely upcoming distribution were classified by more instances. The second reason lies in the nature of the game. Most of the distributions that are less probable to occur have either a very high or a very low score – games with a lot or few trumps and jacks are usually easy to dismiss or clearly winnable. In those cases, it is not so important to calculate a very close estimate of the possible score. The quality of a bidding system is rather given by its ability to decide whether close games can be won.

5 Discarding with KNN

As outlined in Section 3, discarding can also be implemented with KNN. For a fixed trump, all 66 possibilities to discard any two cards out of twelve must be examined. If the trump is not decided yet, there are even 330 possibilities, not considering null games. Although KNN is a pretty fast algorithm, we drop some of these possibilities. First, only those trumps are considered that are still playable given the last bid of the bidding auction. Second, it never makes sense to discard a jack or a trump card. Hence, those possibilities are excluded as well. The remaining possibilities are all classified with our bidding system, yielding a discarding algorithm that only needs 0.1 seconds for discarding and game announcement.

For the discarding system, the knowledge base was changed slightly. On the one hand, the score of each instance was lowered by the value of the skat. On the other hand, the result of the evaluation functions $V^t(s)$ was then increased by the value of the discarded cards. In bidding, an estimation of the points in the skat is necessary, because those points belong to the declarer but are unknown while bidding. That estimation was given naturally by the structure of the knowledge base. For discarding on the other hand, no estimation is necessary as the real point value of the discarded cards is known.

A detailed evaluation of this process was impossible: to discard all 66 possibilities for the 5,000 evaluation games and play the resulting game with every trump would haven taken too long with DDSS. Therefore, we randomly chose 100 games and calculated all possibilities just for those. Two representative games are shown in Fig. 2, both the calculated optimum and the bidders result.

For the upper game, our approach discards exactly the cards calculated as optimal. The lower one does not match the calculated optimum, but it seems that this optimum is more questionable than our result. First, it is definitely better to discard ♢A instead of ♢10, because the declarer will have a higher score – in both cases the remaining card will win the same tricks. Regarding the second row, most skilled players try to get rid of one suit completely as proposed by our algorithm. The questionable calculation is caused by the small number of samples, as already stated by the authors of DDSS. Note that similar errors in our knowledge base distort the bidding systems results as well.

trump: ♡	hand												skat	
bidder	♣J	♠J	◇J	♣7	♡10	♡Q	♡9	♡8	◇A	◇Q	♠10	♠Q		
DDSS	♣J	♠J	◇J	♣7	♡10	♡Q	♡9	♡8	◇A	◇Q	♠10	♠Q		
trump: ♠	**hand**												**skat**	
bidder	♠J	◇J	♣Q	♣7	♠Q	♠9	♠7	◇A	◇10	◇9	♡Q	◇K		
DDSS	♠J	◇J	♣7	♠Q	♠9	♠7	♡Q	◇A	◇K	◇9	♣Q	◇10		

Fig. 2. Discarding with KNN

However, the performance of our discarding system is even better in all regarded games than the bidding system's performance so far. This is mainly due to the changes in the knowledge base.

6 Bidding with the Skat in Mind

The knowledge base for bidding can also be improved in a similar way. It was already mentioned, that the bidder estimates the scores based on *hand* games that are randomly generated. Of course, this does not describe Skat satisfyingly, as most games simply cannot be won without picking up the skat. This causes an underestimation. Therefore, the knowledge base was changed as follows. For all games the system picked up the skat, discarded by use of the discarding system, and the resulting games were then calculated with DDSS. This new score simply replaced the former one in the knowledge base – the features were left unchanged as they describe a hand during bidding sufficiently.

Now complete games were simulated: we assume that the bidder has to bid until its highest possible bid is reached and thus cannot play trumps with a lower value. The games the bidder chose to bid on were calculated (after discarding) by DDSS. The rest of the games (the ones the bidder classified as $\emptyset$) were discarded as well and then calculated by DDSS. If they were not won by DDSS we regarded them as not winnable. Additionally, different values for the threshold th were tested. That value can serve as a risk modulator, the lower it is the riskier the system bids. Figure 3 shows the results of the evaluation.

The overall results are very similar to the ones of our first presented implementation. With 75 %, they are just slightly higher. However, the change to a more realistic scenario made the task a bit harder – the system was not allowed to announce every trump, so an improvement was definitely achieved. Also, the underestimation has turned into the preferred overestimation, as the optimal threshold lies between 61 and 64. In general the range in which good results are obtained is rather large. Everywhere between 52 and 67 more than 70 % of the games are classified correctly, while the percentage of correctly classified winnable or not winnable games differs a lot in this range. This is a perfect precondition for a regulating system for the desired risk.

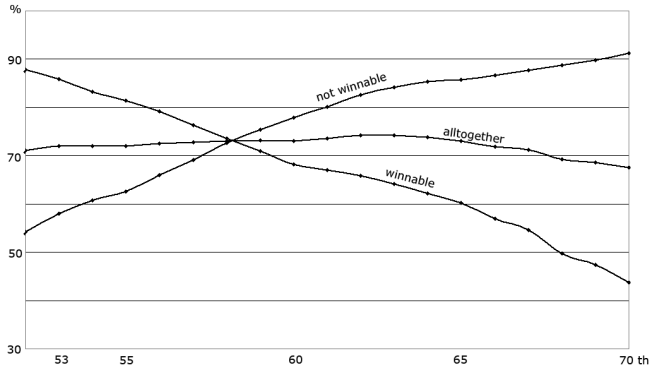

Fig. 3. Evaluation results for bidding

7 Conclusion

Regarding the dimension of imperfect information in a game of Skat, the presented results are impressive. We implemented an efficient bidding and discarding system. First comparisons with Schäfer's bidding system [2] revealed that our implementation detects more of the winnable games, without playing too risky. In general, KNN seem very well suited for tasks with complex separation functions like bidding in skat. Our final system classifies 75 % of all games correctly.

The results may even be improved with a knowledge base created by DDSS using more samples. This is one of the possible improvements planned for the future. Applying the results of the last section to discarding might further improve the discarding systems behavior as well. We also plan to investigate how our system can be further improved by combining it with a Monte Carlo approach. Therefore the features have to be expanded on the opponent's hands as well (that are known for the instances of the knowledge base), and the query has to be sampled. Because KNN is a very time efficient algorithm, the additional computing time should be neglectable.

References

1. Kupferschmid, S., Helmert, M.: A Skat player based on Monte-Carlo simulation. In: Proceedings of the 5th International Conference on Computer and Games. Volume 4630 of LNCS., Springer-Verlag (2007) 135–147
2. Schäfer, J.: Monte-Carlo-Simulationen bei Skat. Bachelor thesis, Otto-von-Guericke-Universität Magdeburg (2005)
3. International Skat Players Association: International Skat and Tournament Order 2007. Web site: `http://www.skatcanada.ca/canada/forms/rules-2007.pdf` (2008)
4. Mitchell, T.M.: Machine Learning. McGraw-Hill (1997)
5. Ginsberg, M.L.: GIB: Steps toward an expert-level Bridge-playing program. In: Proceedings of the 16th International Joint Conference on Artificial Intelligence. (1999) 584–589
6. Frank, I., Basin, D.A.: Search in games with incomplete information: A case study using Bridge card play. Artificial Intelligence **100** (1998) 87–123