A Doppelkopf Player Based on UCT

Silvan Sievers and Malte Helmert

University of Basel, Switzerland
{silvan.sievers,malte.helmert}@unibas.ch

Abstract. We propose doppelkopf, a trick-taking card game with similarities to skat, as a benchmark problem for AI research. While skat has been extensively studied by the AI community in recent years, this is not true for doppelkopf. However, it has a substantially larger state space than skat and a unique key feature which distinguishes it from skat and other card games: players usually do not know with whom they play at the start of a game, figuring out the parties only in the process of playing.

Since its introduction in 2006, the UCT algorithm has been the dominating approach for solving games in AI research. It has notably achieved a playing strength comparable to good human players at playing go, but it has also shown good performance in card games like Klondike solitaire and skat. In this work, we adapt UCT to play doppelkopf and present an algorithm that generates random card assignments, used by the UCT algorithm for sampling. In our experiments, we discuss and evaluate different variants of the UCT algorithm, and we show that players based on UCT improve over simple baseline players and exhibit good card play behavior also when competing with a human player.

1 Introduction

Doppelkopf (literally “double head”) is a trick-taking card game for four players. It is mostly played in Germany, with a popularity only slightly lower than that of *skat*, which has been well-studied by the AI community in recent years [16, 14, 5, 17, 10, 11]. However, to the best of our knowledge, doppelkopf has not been subject to AI research yet, although it has a much larger state space than skat and a unique feature which potentially makes it harder for computers playing it due to increasing uncertainty: at the start of a game, the parties are usually not known to the players until specific cards have been revealed, and hence collaboration at the beginning is difficult and subject to assumptions and inference. This work aims at introducing doppelkopf as a research topic and providing a baseline against which future work can compare.

Upper Confidence Bounds applied to Trees (UCT) [15] is a *Monte Carlo* tree search algorithm which has experienced a lot of success since its first application to the game of go [12]: It has successfully been used for (complete information) General Game Playing [9], and for many scenarios of acting under uncertainty, including the Canadian Traveler’s Problem [8], probabilistic planning [13], Klondike solitaire [4], and multi-player games like hearts, spades, Chinese checkers [23], and skat [19]. Hence, it seems natural to use UCT to establish a baseline computer player for doppelkopf. To be able to use UCT on a full game state space in contrast to the actual belief state space the doppelkopf players are confronted with, we present an algorithm that computes random card assignments consistent with the game history, which can then be used by the UCT algorithm for game simulations. In our experiments, we evaluate different variants of UCT in different settings, showing that players based on UCT improve over baseline players and show good card play behavior also when playing with a human.

2 Doppelkopf

In this section, we present doppelkopf in a short form, discussing only the basic rules. Official rules have only been defined after the foundation of the Deutscher Doppelkopf Verband (German Doppelkopf Association; DDV) in 1982 [1]. Consequently, there is a large pool of house rules used by hobby players. In this work, we exclusively consider the official rules which are used for tournament play. For an exhaustive description of the game, we refer to the first author's Master's thesis [22] or the DDV [1].

2.1 Game Play

Doppelkopf is a trick-taking card game for four players. The deck consists of a double shortened French deck, with a total of 48 cards (the 2s to 8s are removed). This means there are the regular four suits clubs (♣), spades (♠), hearts (♡) and diamonds (♢), each consisting of two aces (A), two tens (10), two kings (K), two queens (Q), two jacks (J) and two nines (9). The card point values are the same as in skat, i.e. an ace is worth 11 points, a ten 10, a king 4, a queen 3, a jack 2, and a nine 0 points, which sums up to a total of 240 card points in a doppelkopf deck. In every game, the *re* party competes against the *kontra* party, (usually) aiming at collecting at least 121 card points.

A game starts by dealing every player 12 cards, the so-called *hand*, which must be hidden from and not communicated to the other players. Before the actual *card play phase* can start, players need to settle the type of the current game in the *game type determination phase*. While there are various *solo game* types where one player forms the re party and plays against the three other players of the kontra party, the most common and default game type is the so-called *normal game*, where two players play against the two others. The chosen game type also dictates how the cards are divided into *suits* and what *rank* each card has in its suit. For the remainder of this section, we will focus on normal games and again refer to the literature for more details about other game types [22, 1].

In a normal game, there is a *trump suit*, consisting of the following cards in descending order: ♡10, ♣Q, ♠Q, ♡Q, ♢Q, ♣J, ♠J, ♡J, ♢J, ♢A, ♢10, ♢K, ♢9. The remaining suits (♣, ♠, ♡) each form a separate non-trump suit, sorted as follows: A, 10, K, 9 (with the exception of ♡, where the tens are part of the trump suit). The parties are as follows: the players holding a ♣Q form the re party, opposing the other players forming the kontra party, and hence the parties are not known to the players at the beginning of a game. If a player p has both ♣Q, p has the option to play a solo or to play a *marriage*, where one of the other three players, according to specific rules, joins the marriage player p to form the re party. As soon as the parties are settled in the case of a marriage, the game continues like a normal game.

The card play rules are independent of the chosen game type and exactly the same as in skat. In short, one player starts by playing any card, thereby settling the suit to be played in this *trick*, which everybody has to follow if possible. The player who played the highest card of that suit (or of the *trump suit*) *wins* the trick, adding its summed card point value to their account, and starts the next trick by playing any card. After completing twelve tricks, the game ends, the winning party and the *game value* are determined (see below), and the next game can start.

In the card play phase (during the first few tricks), independently of the game type, players can make *announcements* to increase the game value. The "basic" announcements (in German: "Ansagen", literally "to announce") are *re* and *kontra*, depending on the announcing player's party. They do not increase the threshold for winning the game, but reveal the announcing player's party to the others. Further announcements (in German: "Absagen", literally "to reject") are *no 90* (*no 60*, *no 30*), which claims that the other party will not manage to win 90 (60, 30) card points, and *schwarz* (literally "black"), which claims that the other party will not win a single trick. If the other party reaches the "rejected goal" nevertheless, the announcing party loses the game. The official rules precisely specify the latest time point in terms of number of cards a player is still holding up to which the player is allowed to make (further) announcements.

2.2 Game Evaluation

The game value is determined in terms of *score points* (not to be confused with card points): $+1$ for winning the game; $+2$ if re (kontra) was announced; $+1$ if the losing party has less than 90 (60, 30) card points or won no trick; $+1$ if no 90 (no 60, no 30, schwarz) was announced; $+1$ if the winning party achieved at least 120 (90, 60, 30) card points against an announcement of no 90 (no 60, no 30, schwarz) of the other party. The players of the winning party are rewarded with the positive game value, the others with the negative game value (zero-sum notation). In case of solo games, the game value is multiplied by 3 for the soloist.

In normal games, there are additional extra score points rewarded as follows: $+1$ for *winning against the elders*, i.e. the kontra party wins; $+1$ for the party *catching a fox*, i.e. winning a trick where an opponent played an $\diamondsuit$A; $+1$ for the party *making a doppelkopf*, i.e. winning a trick worth at least 40 card points; $+1$ for the party winning the last trick if the highest card played in this trick is a *charlie*, i.e. a $\clubsuit$J.

2.3 Discussion

With 48 cards, there are $\binom{48}{12} \cdot \binom{36}{12} \cdot \binom{24}{12} \cdot \binom{12}{12} \approx 2.4 \cdot 10^{26}$ possible card deals. For a fixed deal, the number of possible game states only in terms of cards that have (not) been played is $\sum_{i=0}^{48} 4 \cdot \binom{12}{\lfloor (i+3)/4 \rfloor} \cdot \binom{12}{\lfloor (i+2)/4 \rfloor} \cdot \binom{12}{\lfloor (i+1)/4 \rfloor} \cdot \binom{12}{\lfloor i/4 \rfloor} \approx 2.4 \cdot 10^{13}$. Multiplying the two numbers gives $5.6 \cdot 10^{39}$, which is only a rough upper bound on the size of the card state space, because game states can be consistent with many different card deals. However, this does not consider the various game types, the moves players make in the game determination phase of the game, or the announcement moves. We think that the large state space of doppelkopf, its uncertainty of knowledge about parties and its strategic depth make doppelkopf an interesting benchmark problem for AI research.

3 The UCT Algorithm

In this section, we present the UCT algorithm [15], a state-of-the-art algorithm for many problems of acting under uncertainty, adapted to doppelkopf. We start with a high-level description of the algorithm. While we ideally would like to determine the move which maximizes the expected outcome of the game for the moving player in a given game state, computing all possible card deals consistent with the game history and computing all possible outcomes of a game under a given card assignment is usually infeasible. The UCT algorithm avoids this problem by relying on sampling. More precisely, UCT is a Monte Carlo tree search algorithm which repeatedly simulates the game starting in the current state until a terminal game state is reached, also called performing a *rollout*. To perform such a rollout of the game, the UCT algorithm needs to assume a fixed card deal, i.e. a fixed card assignment of the remaining cards to all other players. We discuss how to compute such a card assignment in more detail in the next section and assume a fixed card assignment for the moment. At the end of a rollout, the outcome of the game is used to compute *UCT rewards* for all players. This information is stored in a *tree* of game states which is incrementally built over the course of performing rollouts and which serves to bias future computations of rollouts. At *any time*, the algorithm can be terminated and the move leading to the successor state with the highest average UCT reward can be returned.

3.1 UCT for Doppelkopf

We now describe the computation of one UCT rollout under a fixed card assignment. Let s_0 denote the current game state for which UCT is queried, let $V^k(s_i)$ denote the *number of rollouts* among the first k rollouts of the UCT algorithm in which state s_i was reached, and let $R^k_j(s_i)$ denote the *average UCT reward* (defined below) obtained by player j in the first k rollouts when completing a rollout from state s_i. Both $V^k(s_i)$ and $R^k_j(s_i)$ are stored in the UCT tree. Each rollout, given the fixed card assignment, starts in state s_0 and iteratively chooses a successor state until a terminal game state is reached. Let s_i be the current state reached in the $(k+1)$st rollout, with n possible successor states $s'_1, \ldots, s'_n$, and let p be the player to move in s_i. UCT favors selecting successors that have led to high UCT rewards for p in previous rollouts (where $R^k_p(s'_j)$ is high) and that have been rarely tried in previous rollouts (where $V^k(s'_j)$ is low). Balancing those two criteria is commonly called the *exploration-exploitation* dilemma, which UCT attempts to solve by treating the decision corresponding to successor selection at state s_i as a *multi-armed bandit problem* and applying the *UCT formula*, which is based on the UCB1 formula introduced by Auer et al. [2] in the context of such multi-armed bandit problems. UCT chooses the successor s'_j which maximizes the UCT formula

$$R^k(s'_j) + C\sqrt{\frac{\log(V^k(s_i))}{V^k(s'_j)}},$$

where C is a bias parameter to guide the amount of exploration. If $V^k(s'_j) = 0$, the value of the *exploration term* (the second term) is considered to be ∞, leading to every successor s'_j being chosen at least once, if s_i is reached at least n times. In our experiments, we use a random successor state from the set of unvisited ones, leading to a variant of UCT which is sometimes called *blind* UCT. After the completion of a rollout, the V and R values of states visited during the rollout are updated.

When reaching a terminal game state, the algorithm has to compute the UCT reward for all players, based on their achieved game score points. This is done by multiplying the game score points with 500 and adding to it, as a bias towards achieving more card points, the card points achieved by the party.[1] We choose 500 to ensure that the card points bias serves a pure tie breaker only, because the maximum of 240 card points can never exceed 500, which is obtained if assuming the smallest possible game value of 1.

UCT comes with theoretical guarantees about convergence and regret bounds: Kocsis and Szepesvári [15] prove that every state is visited infinitely many times, given enough rollouts, and that states which have been unpromising previously are chosen less and less frequently over time, proving that the algorithm eventually converges to a stable policy. Furthermore, they prove that the average UCT reward of every state lies in the interval of its expectation plus minus the exploration term, which grows logarithmically in the number of visits of the state, i.e. the regret of the policy UCT converges to is bounded from the optimal policy logarithmically in the number of rollouts.

3.2 Variants

Recent work in the area of General Game Playing discusses modeling the belief state space of incomplete information games and using classical complete information algorithms such as Minimax or UCT on the actual states corresponding to a belief state [20, 7, 21]. While we stick to the approach of instantiating belief states into full game states and then using UCT on the complete information state space — which proved to be very successful in various applications [8, 13, 4, 23, 19]— we propose two *versions* of the UCT algorithm to vary the number of card assignments used. The first version, called *ensemble-UCT*, performs several regular UCT computations, each with a different card assignment, fixed over all rollouts of that computation. Hence, it constructs a different tree for every such UCT computation and in the end chooses the action which leads to the state maximizing the average UCT reward, averaged over all UCT computations. The second version, called *single-UCT*, computes a new card assignment for every rollout. As a consequence, the constructed tree can contain states which are not consistent across different rollouts. In a given rollout, only successors consistent with the current card assignment are considered for selection. This leads to fewer parts of the tree being reusable over different rollouts, but avoids the potential problem of computing only a few different card assignments as in the ensemble-UCT version.

Furthermore, we consider two variants of constructing the tree over the course of performing rollouts that differ in how previously unseen states are handled. The first variant adds a node for *every* new state encountered during a rollout to the tree. The second variant only adds the *first* new state encountered and continues the remainder of the rollout as a so-called *Monte Carlo simulation*, which consists of random successor selection until a terminal game state is reached. Note that this behavior does not differ

[1] We also experimented using the card points achieved only by the player and obtained results with no significant difference.

from the first variant, which also chooses random successors when encountering a state with previously unseen successors. The difference only lies in the amount of statistics recorded for states encountered during rollouts. With the first variant, the constructed tree grows quickly, including many nodes in the tree that will potentially not be visited again, especially if they led to “bad” results in the rollout they were encountered. Still, the more information available, the more quickly the UCT algorithm converges. With the second variant, the tree grows more slowly, only including information about states that have a high probability of being revisited in future rollouts. We suspect that not performing a Monte Carlo simulation should be beneficial particularly with the ensemble-UCT version, because it uses the same card assignment over many rollouts and is hence expected to obtain similar results in different rollouts for the same state. With the single-UCT version, different rollouts may lead to very different outcomes even for the same state.

4 The Card Assignment Problem

We now consider the *card assignment problem* (CAP): in a game state s with player p to move, assign all remaining cards to all other players such that the card assignment is *consistent* with the game history. A card assignment is consistent if it respects all information about the other players' hands available to p. Our goal when solving the CAP is to compute a solution uniformly at random to avoid generating any bias towards specific card assignments in the computation of the UCT algorithm.

In state s, the following information about other players' hands is available to p: if a player p' is playing a marriage[2] and did not play one or both of their ♣Q yet, p' still needs to have both or one; re players not having played their ♣Q yet must still hold it and kontra players cannot hold a ♣Q, where in both cases many scenarios can lead to knowing a player's party; all players who could not follow suit at some point in the game cannot have any card of that suit.

To assign cards uniformly at random, we have to be able to compute the exact number of possible consistent card assignments for arbitrary states. Then, taking into account the number of consistent card assignments before and after hypothetically assigning a card to player, we can compute the exact probability for this specific assignment. However, computing the number of solutions for the CAP, i.e. computing the number of consistent card assignments, is #P-hard:[3] the CAP can be formulated as a *matching problem* in graph theory (see e.g. [6]), where solving the CAP corresponds to finding a perfect bipartite matching, or more generally as a *CSP* (see e.g. [18]). Computing the number of perfect matchings in a bipartite graph (#PBM) is #P-complete, because it is closely connected to the #P-complete problem of computing the permanent of 0-1-matrices [24], and counting the solutions of a CSP (#CSP) is a straightforward generalization of #PBM. For #PBM, there exists a fully polynomial time randomized approximation scheme [3]. However, using this approximation seems to demand an infeasible amount of computation, considering that we would need to solve the CAP up

[2] In specific situations in the game type determination phase, it is also possible to *infer* that a player wanted to play a marriage.

[3] Informally, #P is the class of counting problems associated with decision problems in NP.

to four times for every remaining card to be assigned, and that we would need to compute a new assignment of all remaining cards for every rollout if using the single-UCT version. We hence propose our own algorithm to approximate a uniformly random card assignment in the following.

The algorithm first computes the set of cards that every player can potentially have without violating consistency. It then performs the following steps, every time starting over again after successfully assigning one or more cards to a player: if there is a card that can only be assigned to one player, assign it to that player; if there is a player that needs exactly as many cards assigned as there are possible cards it can have, assign all those cards to that player; if there is a player that needs one ♣Q, assign one to that player; otherwise assign an arbitrary of the remaining cards to an arbitrary player that can have it.

It is easy to see that the algorithm can only generate consistent card assignments, because it never assigns a card to player that is not allowed to have it in a consistent card assignment. We further argue that there exists a possible card-to-player assignment of all remaining cards in every iteration of the algorithm: if a card cannot be assigned to any player, then there must have been an earlier iteration in which that card could only have been assigned to one player and the algorithm would have chosen that assignment. Because the algorithm assigns at least one card in every iteration, it terminates after at most as many iterations as there are cards left to be assigned initially.

Our algorithm does not generate consistent card assignments uniformly at random for several reasons: first, the algorithm does not consider the number of possible card assignments when randomly choosing a card and a player to assign to. Second, it does not consider the number of card slots of the player, hence treating assigning a card to a player with one open slot as likely as assigning it to a player with many open slots. Third, prioritizing the assignment of a ♣Q to a player does not consider the probability of that player getting a ♣Q without enforcing it.

5 Experiments

In this section, we report results for several variants of the UCT algorithm for doppelkopf. While there are many commercial doppelkopf programs and the open source program FreeDoko[4] which comes with various types of configurable computer players (e.g. using simulations, search trees, or heuristics), it is a technically challenging task to evaluate these different programs together with our algorithm due to the lack of a common framework. We hence evaluate the UCT players in competition against each other and fill up empty player spots with random players. Random players do not take part in the game determination phase, do not make announcements and play random cards in every trick. They can be seen as "dummy players" which do not influence the game.

Our general experimental setup is as follows: we compare two different UCT players by letting them play together with two identical random players, playing 1000 games with random card deals. We repeat those 1000 games in *every possible permutation* of

[4] http://free-doko.sourceforge.net/en/FreeDoko.html

player positions, so that every player plays every hand on every position once. The UCT algorithm generally computes 10000 rollouts every time it is queried for a game state, independent of the UCT version used. We evaluate a player's performance as the average score points per game in the 95% confidence interval. We report the results achieved by each UCT player and, in parenthesis, by both random players. The best result of every comparison is highlighted in bold. Note that with overlapping confidence intervals, the results are not always statistically significant. More precise results could be obtained by increasing the number of games played.[5]

5.1 Exploration in the UCT Algorithm

In a first series of experiments, we investigate the influence of exploration in both the ensemble-UCT and the single-UCT versions. Simultaneously, we test different combinations of the number of performed UCT computations X and the number of rollouts Y performed in each such computation for the ensemble-UCT version (denoted X/Y in the following). We disable performing Monte Carlo simulations and discuss their influence below.

Table 1. Comparison of different values for the exploration bias C for different configurations of the ensemble-UCT version.

ensemble-UCT (5/2000)							
$C = 6000$	vs.	$C = 8000$	1.48 ± 0.07	vs.	$\mathbf{1.90 \pm 0.08}$	(vs.	-1.69 ± 0.05)
$C = 8000$	vs.	$C = 12000$	1.42 ± 0.08	vs.	$\mathbf{2.07 \pm 0.10}$	(vs.	-1.74 ± 0.05)
$C = 12000$	vs.	$C = 16000$	1.65 ± 0.10	vs.	$\mathbf{1.83 \pm 0.12}$	(vs.	-1.74 ± 0.05)
$C = 16000$	vs.	$C = 24000$	$\mathbf{1.63 \pm 0.12}$	vs.	1.41 ± 0.13	(vs.	-1.52 ± 0.06)
$C = 24000$	vs.	$C = 32000$	$\mathbf{1.35 \pm 0.13}$	vs.	1.20 ± 0.13	(vs.	-1.27 ± 0.06)
ensemble-UCT (10/1000)							
$C = 6000$	vs.	$C = 8000$	1.49 ± 0.06	vs.	$\mathbf{1.94 \pm 0.07}$	(vs.	-1.72 ± 0.05)
$C = 8000$	vs.	$C = 12000$	1.49 ± 0.07	vs.	$\mathbf{2.16 \pm 0.09}$	(vs.	-1.83 ± 0.05)
$C = 12000$	vs.	$C = 16000$	1.82 ± 0.10	vs.	$\mathbf{2.01 \pm 0.11}$	(vs.	-1.92 ± 0.05)
$C = 16000$	vs.	$C = 24000$	$\mathbf{1.92 \pm 0.11}$	vs.	1.79 ± 0.12	(vs.	-1.86 ± 0.05)
$C = 24000$	vs.	$C = 32000$	$\mathbf{1.86 \pm 0.12}$	vs.	1.66 ± 0.12	(vs.	-1.76 ± 0.06)
ensemble-UCT (20/500)							
$C = 6000$	vs.	$C = 8000$	1.52 ± 0.06	vs.	$\mathbf{1.95 \pm 0.07}$	(vs.	-1.73 ± 0.05)
$C = 8000$	vs.	$C = 12000$	1.65 ± 0.07	vs.	$\mathbf{2.08 \pm 0.09}$	(vs.	-1.86 ± 0.05)
$C = 12000$	vs.	$C = 16000$	1.88 ± 0.09	vs.	$\mathbf{1.91 \pm 0.10}$	(vs.	-1.90 ± 0.05)
$C = 16000$	vs.	$C = 24000$	$\mathbf{1.93 \pm 0.10}$	vs.	1.75 ± 0.10	(vs.	-1.84 ± 0.05)
$C = 24000$	vs.	$C = 32000$	$\mathbf{1.93 \pm 0.10}$	vs.	1.59 ± 0.10	(vs.	-1.76 ± 0.05)

Table 1 shows the results of the ensemble-UCT version. On a pairwise basis, we compare using different values for the exploration bias C, each for the three combinations 5/2000 (first block), 10/1000 (second block), and 20/500 (third block). The first observation is that the results are consistent across all configurations: With increasing

[5] Computing a set of 1000 games with two UCT players using the ensemble-UCT version takes roughly 19.5h in average to complete. With the single-UCT version, the average is roughly 46h, due to the high number of card assignments which need to be computed.

C, performance first increases up to a sweet-spot approximately around $C = 16000$ and then decreases. Considering that the UCT reward can theoretically range from -16740 to $+16740$ (assuming a solo game with the highest possible value of 11, not including irrational counter-announcements), and most often will range from -2000 to $+2000$ (assuming an average game value of 4), a value of $C = 16000$, multiplied with the square root term of the UCT formula (ranging between 0 and 1), seems to be appropriate to strongly favor exploration at the beginning, reducing exploration more and more as the number of rollouts increases.

Next, we compare the three combinations with a fixed C against each other. Table 2 shows that the combination 10/1000 achieves the best performance. Apparently, there is a trade-off between the number of different card assignments considered and the number of rollouts performed with a fixed card assignment. Both using too few different card assignments and performing too few rollouts with each card assignment hurts the quality of the UCT computation.

Table 2. Comparison of different combinations of the number of UCT computations and rollouts for the ensemble-UCT version.

ensemble-UCT (C=16000)							
5/2000	vs.	10/1000	1.67 ± 0.12	vs.	$\mathbf{1.83 \pm 0.11}$	(vs.	-1.75 ± 0.05)
10/1000	vs.	20/500	$\mathbf{2.10 \pm 0.11}$	vs.	1.70 ± 0.10	(vs.	-1.90 ± 0.05)

We now turn our attention to results of the single-UCT version displayed in Table 3. We observe a similar trend for changing values of approximately C, however the best performance is achieved with a lower value of $C = 8000$. Furthermore, the performance of the players using the single-UCT version compared to the baseline players is weaker than with the ensemble-UCT version. This lower absolute performance also explains why choosing a lower value for C is better with the single-UCT version. We will compare the two UCT versions directly against each other below.

Table 3. Comparison of different values for the exploration bias C for the single-UCT version.

single-UCT							
$C = 2000$	vs.	$C = 4000$	0.26 ± 0.06	vs.	$\mathbf{1.18 \pm 0.08}$	(vs.	-0.72 ± 0.05)
$C = 4000$	vs.	$C = 6000$	0.37 ± 0.07	vs.	$\mathbf{1.32 \pm 0.09}$	(vs.	-0.85 ± 0.05)
$C = 6000$	vs.	$C = 8000$	0.53 ± 0.07	vs.	$\mathbf{0.85 \pm 0.09}$	(vs.	-0.69 ± 0.05)
$C = 8000$	vs.	$C = 10000$	$\mathbf{0.52 \pm 0.08}$	vs.	0.47 ± 0.09	(vs.	-0.50 ± 0.05)
$C = 10000$	vs.	$C = 12000$	$\mathbf{0.33 \pm 0.08}$	vs.	0.33 ± 0.09	(vs.	-0.33 ± 0.04)
$C = 12000$	vs.	$C = 14000$	$\mathbf{0.30 \pm 0.09}$	vs.	0.05 ± 0.09	(vs.	-0.18 ± 0.05)

For the remainder of this section, we choose the combination 10/1000 and $C = 16000$ for the ensemble-UCT version and $C = 8000$ for the single-UCT version.

5.2 Influence of Announcements and Monte Carlo Simulations

In the following, we discuss the influence of two other parameters. We start by investigating the influence of making announcements by comparing a UCT player who is allowed to make announcements against one who is not. Table 4 shows the comparison for both UCT versions. We observe that for both UCT versions, UCT players clearly profit from being allowed to make announcements. This means that on average, they win more games than they lose if they make an announcement. Finding the right amount of announcement making is a very important way of increasing the score points gained on average. Also on a human level of playing, this is crucial for top performance.

Table 4. Comparison of allowed and forbidden announcing for both UCT versions.

ensemble-UCT (10/1000, C=16000)							
Announcing	vs.	no announcing	$\mathbf{1.70 \pm 0.07}$	vs.	0.79 ± 0.05	(vs.	-1.25 ± 0.04)
single-UCT (C=8000)							
Announcing	vs.	no announcing	$\mathbf{0.48 \pm 0.06}$	vs.	0.19 ± 0.05	(vs.	-0.33 ± 0.04)

Second, we investigate the influence of performing a Monte Carlo simulation when reaching a new state, adding only one new state rather than all states encountered during rollouts to the tree. Table 5 shows the comparison for both UCT versions. We observe that for the ensemble-UCT version, not performing a Monte Carlo simulation achieves a significantly better result than performing a simulation, while the opposite is true for the single-UCT version. This confirms our assumption that the ensemble-UCT version, due to the fixed card assignment for every single UCT computation, particularly profits from not performing a simulation but recording all information gained during rollouts. Using the single-UCT version, information gained in previous rollouts may be misleading in rollouts with different card assignments, hence storing and reusing all information can be harmful.

Table 5. Comparison of using and not using a Monte Carlo simulation for both UCT versions.

ensemble-UCT (10/1000, C=16000)							
No MC simulation	vs.	MC simulation	$\mathbf{2.15 \pm 0.11}$	vs.	1.73 ± 0.09	(vs.	-1.94 ± 0.05)
single-UCT (C=8000)							
No MC simulation	vs.	MC simulation	0.34 ± 0.08	vs.	$\mathbf{0.85 \pm 0.08}$	(vs.	-0.59 ± 0.05)

5.3 Ensemble-UCT versus Single-UCT

Our next experiment compares the two UCT versions in their best configuration (ensemble-UCT: 10/1000, $C = 16000$, no Monte Carlo simulation; single-UCT: $C =$

8000, Monte Carlo simulation) directly against each other. The ensemble-UCT version achieves a score points average of 4.52 ± 0.11, the single-UCT version -1.25 ± 0.08, and the random players -1.63 ± 0.05. Hence our previous observation is not only confirmed, but the performance of the single-UCT version drops even below 0, and remains only slightly above the baseline players' performance. We conclude that using a new card assignment for every rollout leads to less informed UCT trees, probably caused by the fact that rollouts may be incompatible to each other, frequently preventing reusing all information across rollouts.

5.4 Game Analysis Against a Human

Finally, we evaluate the ensemble-UCT version in a setup with a human and two random players, playing two tournaments consisting of 24 games each. Note that we cannot repeat these games in all permutations because a human cannot easily forget previously played hands. Furthermore, playing 24 games is a very low amount to draw conclusions from. Hence the results are not significant, but the games still serve as a basis for investigating the playing behavior of the UCT player in the following. The results of the first (second) tournament are as follows: the human player achieves 43 (15) score points, the UCT player -9 (7), and the random players -15 (-35) and -19 (13), respectively.

We observe several trends in the playing behavior of the UCT player p: first, p plays many solo games: 7 (9) in the first (second) tournament, winning only 4 (5) of them. While the absolute value of score points achieved in those games is positive ($+3$ and $+15$), it is a lower average per game than what p (probably) could have achieved by playing a normal game. Analyzing the hands p decided to play a solo with for their use in a game with four humans, we think that 4 (4) hands have no chances of being won in the first (second) tournament, 2 (3) have borderline chances (e.g. depend on p being the starting player and on a "good" card distribution to other players), and 1 (2) have a good chance of being won or are clear solo hands. In nearly all of the cases, the hands are excellent for a normal game, and especially with such hands, playing a solo only makes sense if there is a very high chance of winning.

Second, in normal games, p never makes an announcement unless the opposing party already made an announcement. All of the games with such a "counter-announcement" are lost for p, leading to a loss of 10 score points in each of the tournaments. However, p always announces re when playing a solo, which is reasonable.

Third, we observe that p plays the stronger the earlier it knows the parties (e.g. in all marriage games) – which should not be surprising, as this usually reduces the amount of reasonable card moves drastically. Also generally, p's playing strength increases over the course of a game, with fewer remaining options to play. For example, it won the last trick with a charlie several times, scoring an extra score point. This increasing playing strength also (partly) explains the less informed decisions about solo playing and the (missing) announcement making, both taking place at the beginning of a game.

5.5 Discussion

Our experiments show that using an ensemble of UCT computations has clear benefits over only performing a single UCT computation which uses different instantiations of the current belief state in every rollout. While a player based on the ensemble-UCT version shows good card play performance, the ability to correctly evaluate a hand for solo play and to decide whether to make an announcement or not is less developed. A possibility of enhancing the performance of such players with respect to hand evaluation would be to use a separate algorithm for this purpose, as it has e.g. been done in skat [14]. An algorithm for hand evaluation could profit from analyzing all cards simultaneously, in the context of a given game type, rather than ranking every possible move by estimating the outcome of the game when making this particular move next.

More generally, there are other possibilities of improving our current UCT players. First, analyzing the bias of the card assignment algorithm could help in improving the algorithm to come closer to uniformly at random generating card assignments, which in turn could have a positive impact on the performance of UCT. Second, and this seems to be very important, domain specific knowledge could help in the simulation phase of a rollout: rather than randomly choosing a successor, using a heuristic approach such as a rule based successor selection could greatly improve the quality of rollouts, especially of the first few hundred rollouts of a UCT computation, where little or no information from previous rollouts is available. In particular, this would accelerate the convergence of the UCT algorithm. Third, the computation of UCT rollouts currently underlies the assumption that the other players act similar to the UCT player itself, i.e. the decision making for other players is the same. This assumption could be dropped and decision making for other players could be replaced by heuristic based or even random successor selection. Fourth, a UCT player could keep relevant parts of the tree(s) constructed in previous UCT computations for the next move it will be queried for, thus starting the new UCT computation with an already initialized tree. This would potentially accelerate the convergence of the UCT computation.

6 Conclusion

We introduced doppelkopf as a benchmark problem for AI research. Although the game play is similar as in skat, a well-established game in the AI community, doppelkopf has a much larger state space and more strategical playing depth. As a baseline for future research, we adapted the UCT algorithm, a state-of-the-art approach for many other scenarios of acting under uncertainty. We discussed the problem of uniformly at random generating consistent card assignments, which are required to compute UCT rollouts under full information. Because this computation is inherently hard, we presented our own algorithm to approximate the uniformly-at-random computation. We experimented with several variants of the UCT algorithm and obtained good results against baseline players. While hand evaluation, required for solo play decisions and announcement making, tends to be overly optimistic, UCT based players showed good card play skills.

We presented several ideas for improving our current UCT players in future work. Apart from those improvements, we would like to compare our UCT players against computer players of other types. A first step towards this end could be to replace the random players by rule based or other heuristic players. Second, more importantly, our implementation of UCT players could be integrated with other existing doppelkopf systems such as the open source FreeDoko program.

References

1. Deutscher Doppelkopf Verband, http://www.doko-verband.de, [Online; in German; accessed 2015-04-28]
2. Auer, P., Cesa-Bianchi, N., Fischer, P.: Finite-time analysis of the multiarmed bandit problem. Machine Learning 47, 235–256 (May 2002)
3. Bezáková, I., Štefankovič, D., Vazirani, V.V., Vigoda, E.: Accelerating Simulated Annealing for the Permanent and Combinatorial Counting Problems. In: Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithms. pp. 900–907. SODA '06, ACM (2006)
4. Bjarnason, R., Fern, A., Tadepalli, P.: Lower bounding Klondike solitaire with Monte-Carlo planning. In: Gerevini, A., Howe, A., Cesta, A., Refanidis, I. (eds.) Proceedings of the Nineteenth International Conference on Automated Planning and Scheduling (ICAPS 2009). pp. 26–33. AAAI Press (2009)
5. Buro, M., Long, J.R., Furtak, T., Sturtevant, N.: Improving state evaluation, inference, and search in trick-based card games. In: Boutilier, C. (ed.) Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI 2009). pp. 1407–1413 (2009)
6. Cormen, T.H., Leiserson, C.E., Rivest, R.L.: Introduction to Algorithms. The MIT Press (1990)
7. Edelkamp, S., Federholzner, T., Kissmann, P.: Searching with partial belief states in general games with incomplete information. In: Glimm, B., Krüger, A. (eds.) Proceedings of the 35th Annual German Conference on Artificial Intelligence (KI 2012). Lecture Notes in Artificial Intelligence, vol. 7526, pp. 25–36. Springer-Verlag (2012)
8. Eyerich, P., Keller, T., Helmert, M.: High-quality policies for the Canadian traveler's problem. In: Fox, M., Poole, D. (eds.) Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence (AAAI 2010). pp. 51–58. AAAI Press (2010)
9. Finnsson, H., Björnsson, Y.: Simulation-based approach to general game playing. In: Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence (AAAI 2008). pp. 259–264. AAAI Press (2008)
10. Furtak, T., Buro, M.: Using payoff-similarity to speed up search. In: Walsh, T. (ed.) Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI 2011). pp. 534–539 (2011)
11. Furtak, T., Buro, M.: Recursive Monte Carlo search for imperfect information games. In: 2013 IEEE Conference on Computational Intelligence in Games (CIG), Niagara Falls, ON, Canada, August 11–13, 2013. pp. 1–8. IEEE (2013)
12. Gelly, S., Wang, Y., Munos, R., Teytaud, O.: Modification of UCT with Patterns in Monte-Carlo Go. Tech. Rep. 6062, INRIA (November 2006)
13. Keller, T., Eyerich, P.: PROST: Probabilistic planning based on UCT. In: McCluskey, L., Williams, B., Silva, J.R., Bonet, B. (eds.) Proceedings of the Twenty-Second International Conference on Automated Planning and Scheduling (ICAPS 2012). pp. 119–127. AAAI Press (2012)
14. Keller, T., Kupferschmid, S.: Automatic bidding for the game of skat. In: Dengel, J., Berns, K., Breuel, T.M., Bomarius, F., Roth-Berghofer, T. (eds.) Proceedings of the 31st Annual German Conference on Artificial Intelligence (KI 2008). Lecture Notes in Artificial Intelligence, vol. 5243, pp. 95–102. Springer-Verlag (2008)
15. Kocsis, L., Szepesvári, C.: Bandit based Monte-Carlo planning. In: Fürnkranz, J., Scheffer, T., Spiliopoulou, M. (eds.) Proceedings of the 17th European Conference on Machine Learning (ECML 2006). Lecture Notes in Computer Science, vol. 4212, pp. 282–293. Springer-Verlag (2006)

16. Kupferschmid, S., Helmert, M.: A Skat player based on Monte Carlo simulation. In: Proceedings of the Fifth International Conference on Computers and Games (CG 2006). pp. 135–147 (2006)
17. Long, J.R., Buro, M.: Real-time opponent modeling in trick-taking card games. In: Walsh, T. (ed.) Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI 2011). pp. 617–622 (2011)
18. Russell, S., Norvig, P.: Artificial Intelligence — A Modern Approach. Prentice Hall (2003)
19. Schäfer, J.: The UCT Algorithm Applied to Games with Imperfect Information. Master's thesis, Otto-von-Guericke-Universität Magdeburg (July 2008)
20. Schofield, M.J., Cerexhe, T.J., Thielscher, M.: Hyperplay: A solution to general game playing with imperfect information. In: Hoffmann, J., Selman, B. (eds.) Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence (AAAI 2012). pp. 1606–1612. AAAI Press (2012)
21. Schofield, M.J., Thielscher, M.: Lifting model sampling for general game playing to incomplete-information models. In: Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence (AAAI 2015). pp. 3585–3591. AAAI Press (2015)
22. Sievers, S.: Implementation of the UCT Algorithm for Doppelkopf. Master's thesis, University of Freiburg, Germany (April 2012)
23. Sturtevant, N.R.: An analysis of UCT in multi-player games. In: van den Herik, H.J., Xu, X., Ma, Z., Winands, M.H.M. (eds.) Proceedings of the 6th International Conference on Computers and Games (CG 2008). pp. 37–49. Springer (2008)
24. Valiant, L.G.: The complexity of computing the permanent. Theoretical Computer Science 8, 189–201 (1979)