Artificial Intelligence for the Card Game Doppelkopf in FreeDoko

Introduction to Doppelkopf

Doppelkopf is a team game where each team normally consists of two players. The most distinguishing feature of the game is that the actual pairing is not known from the start, which is what makes the game interesting for most players.

The deck of cards consists of either 48 or 40 cards, with...

* 8 Nines worth 0 points each
* 8 Tens worth 10 points each
* 8 Jacks worth 2 points each
* 8 Queens worth 3 points each
* 8 Kings worth 4 points each, and
* 8 Aces worth 11 points each

... and with each group of 8 cards consisting of 2 cards from each suit: Diamonds, Hearts, Spades and Clubs. Each card exists twice in the deck (which leads to the name Doppelkopf) resulting in a total number of 240 points. In the following explanation, the more common 48-card version is assumed. The rules for the 40-card variant are the same, the only difference is that the Nines are missing.

In every game, there exist two parties, called re and contra. To win, the re-party must make 121 points or more; contra wins when re fails to do so.

[http://en.wikipedia.org/wiki/Doppelkopf]

Complexity of Artificial Intelligence in Doppelkopf

This chapter will give a short introduction to the challenges an artificial intelligence has to master.

For estimation of the complexity of a typical Doppelkopf game we will make the following assumption:
Each player beside the start player of a trick can choose approximately one of round(Cards on Hand / 3 + 0.3) cards which are allowed to be played for the trick. Based on this the complexity for a Doppelkopf game can be calculated:

Trick	Cards Player 1	Cards Player 2	Cards Player 3	Cards Player 4	Number of possible tricks
1	12	4	4	4	768
2	11	4	4	4	704
3	10	4	4	4	640
4	9	3	3	3	243
5	8	3	3	3	216
6	7	3	3	3	189
7	6	2	2	2	84
8	5	2	2	2	40
9	4	2	2	2	32
10	3	1	1	1	3
11	2	1	1	1	2
12	1	1	1	1	1

Based on this just the calculation of all possible first two trick combinations would create a game tree amounting to 537,600 different tricks. Calculating the fifth and sixth trick still amounts to 40,824 different possibilities.

Based on the experience and used algorithms in FreeDoko it is possible to calculate up to 10,000 tricks on typical hardware in a for the human player acceptable time.
This easily shows that even on the theoretical base it is only possible to calculate the whole game from the eights trick onwards.
In trick seven it is possible to take a look for one trick into the future.
For tricks one to six it is only possible to calculate the current trick.

To further increase the problems in this respect, the above shown table is only valid if the cards of all players are known. For an artificial intelligence, which is not cheating, this is not true in most game situations. In this case the number of calculated cards one player can have is much larger then the number used for the calculations in above table.

This situation creates a large problem because algorithms, which calculate a game tree, depend

strongly on the calculation of future tricks in their quality.
If only the current trick is calculated only the best card for this trick is taken. But in many situations the best card for the current trick is not the best card for the game. E.g. the current trick can only be won with a high trump card, but with this card already one trick later much more points could be won.
This shows that game tree algorithms cannot determine even a nearly optimal card for a trick. So there was the need to implement another approach.

A second reason was also that the usage of of game tree algorithms, even when calculating only 10,000 tricks, slows the game play down. Where the time for the game tree calculation is acceptable for one trick, the sum of those calculations for a whole game would not be a acceptable for a human player.

Based on this in FreeDoko heuristics were introduced to improve the decisions and needed time of the artificial intelligence for typical game play situations.

What is a Heuristic?

First let us define what a heuristic is.
Generally a heuristic is a method to help solve a problem, commonly informal. It is particularly used for a method that often rapidly leads to a solution that is usually reasonably close to the best possible answer. Heuristics are "rules of thumb", educated guesses, intuitive judgments or simply common sense.

In computer science, a **heuristic** is a technique designed to solve a problem that ignores whether the solution can be proven to be correct, but which usually produces a good solution or solves a simpler problem that contains or intersects with the solution of the more complex problem.

Heuristics are intended to gain computational performance or conceptual simplicity, potentially at the cost of accuracy or precision.

[http://en.wikipedia.org/wiki/Heuristic]

Based on this let us take a look at one of the simpler heuristics in FreeDoko: PlayAce
This one can be summarized as follows: When serving a trick play a non trump ace of the color with most cards on the hands of the other players.
When doing this consider also that the color was not yet jabbed by the opposing team and that at least enough cards of this color are distributed on the other players that each could have at least one card to follow suit for this trick.

So basically a heuristic consists out of some general conditions in which situation it should be used (above e.g. when serving a trick). Then which card should be used (the color ace with the most cards of the color still in the game) and some guards that the card is really sensible (the color was not already jabbed by other team and at least one card of the color for each player is still available in the game).

So let us take a look how often heuristics can be used in a program like FreeDoko for Doppelkopf.

None of the heuristics implemented in FreeDoko is used for more than 10% of all tricks. But the number of implemented heuristics brings it to a total of only 32% where still no heuristic can be used and by this game tree algorithms are the only option.

No. of Games	**100**	**300**	**500**	**550**
Game tree	32%	31%	32%	32%
Only Valid Card	24%	24%	24%	24%
Serve Color Trick	8%	8%	8%	8%
Best Winning Card	6%	6%	6%	6%
Choose For Color Trick	6%	6%	6%	6%
Play Ace	5%	5%	5%	5%
Play low high trumps	4%	4%	4%	4%
Serve Trump Trick	3%	3%	3%	3%
Choose Pfund	2%	2%	2%	2%
Other	10%	11%	10%	10%

Please note that playing the only valid card to follow suite is not thought of as a heuristic as this is just enforced behaviour by the rules of the game itself (25%).
In the last three tricks also no heuristics are used at all, because in the end game the selection of the right card is much more important and the different ai-type algorithms have a chance to calculate the whole remaining game. So here either the only valid card to follow suite is played or one of the different ai-type algorithms is used (10%).

In FreeDoko there is still the possibility to reduce the number of situations where no heuristic is available. For this with each new release some more heuristics are implemented for the artificial intelligence. But the already implemented heuristics are the ones used most often.

Beyond Heuristics

As shown above for the 24,000 cards in 500 games approximately 8,000 are still calculated via game tree methods.
So this chapter should give an overview over the three different approaches for a game tree calculation used in FreeDoko. Beside game tree there is one further approach to just take random cards from a set of preselected valid cards. This is the fourth implemented approach in FreeDoko.

The first approach is just the one discussed at the beginning of this paper. This one is named GameTree in FreeDoko and just calculates all possible tricks for the current game. By this the result is always the best card that can be found. But as it always calculates at least one trick it is also the most time consuming version and takes sometimes longer than the user accepts.
The second approach is based on GameTree but instead of calculating all possibilities first the available heuristics will be tried to find out if there is a good card instead of calculating each best card. This reduces the time needed by some degree but also the results are slightly reduced in quality.
The next is named VirtualGames and uses virtual players to simulate the game on the current knowledge. This reduces the needed time even further but also reduces the quality of the result further because the partly uncertain knowledge of the game and card distribution determines directly the quality of the result.
The forth version is the fasted, but also with the worst result normally. Here just a number of completely random games is played based on the possible cards of each player. This approach is named MonteCarlo in FreeDoko.

Based on the experience GameTree with Heuristics and VirtualGames calculate always 10,000 tricks, where MonteCarlo calculates 10,000 games.
Based on those settings the following table shows the game points for the four types:

- GameTree with Heuristics
- VirtualGames
- MonteCarlo
- Random cards (with just a small pre filtering of the cards allowed for the current trick)

playing against each other.

No. of Games	**100**	**300**	**500**	**550**
GameTree	186	523	876	995
VirtualGames	143	492	807	901
MonteCarlo	159	454	803	839
Random	156	385	700	775

The results are as expected by showing that GameTree with Heuristics has the best result and Random has the worst result.

Based on this and the calculation time needed by each approach the following usage of the algorithms was decided to be implemented for the artificial intelligence in FreeDoko:
For the first two tricks MonteCarlo is used as the other methods would not calculate more than the current trick and would need much more time.

For the tricks 3 to 8 then VirtualGames is used.
Finally for the tricks 9 to 12 GameTree with Heuristics is used. As this approach will have the best result which is most important for end game phase.

Cards Information

As we have seen above, important for the heuristics is the knowledge of the hands of the other players. The cards information in FreeDoko are processed in three parts:

1. update the certain information
2. change the cards weighting
3. estimate the hands.

Update the certain information

On the first view, the cards information known for certain is simple: If a player does not serve a color (e.g. club), then he does not have any cards of club. And if a player announces re / contra, he does / does not have a club queen.

But we have to look further and check, what information of one player influence the information of another player. E.g. if player 1 (Mara), player 2 (Gerd) and player 3 (Erika) does not have club, player 4 (Sven) must have the remaining club cards. And again we have to look further: If Sven does only have 3 remaining cards and there are 2 cards of club left, he can have at max 1 spade. Assume further, that Mara and Gerd do not have spade and there remain two ten and one king of spade. Since Mara, Gerd and Sven together have only at max 1 spade, Erika must have at least all two remaining spade, but can also have all three. But we know, that Erika must have one ten of spade.

In this example we see, that one new information must be checked against all known information and if we get another change in the information, we must recheck again.
In FreeDoko we have one counter for each card and one for each card color (where trump is a stand-alone-color). We count how many cards / cards of a card color a player has played, still must have (at minimum) and still can have (at maximum).

Change the cards weighting

In the next step, we weight the cards according to the gameplay. If p.e. the first player does not start with a color ace, we assume, that he does not have any (add a negative weighting). And if the last player jabs a trick with a heart queen although a club jack would be sufficient, we assume, he will not have any of club jack and diamond queen (add a negative weighting). In FreeDoko we use heuristics for the weighting, they can be seen as a counterpart of the heuristics described above.
Note: the weighting also depends on the teams of the players (e.g.. whether a ten of diamond is given to the own or the opposite team). The difficulty here is, that the team information is not known from the beginning and is estimated from the cards played, also.

Estimate the hands

For each player we start with the cards he must have. Then we add the cards he can have with the greatest weighting into his estimated hand. Further we add the cards for which he has the greatest weighing according to the other players (this ensures, that each card is distributed to a player). Afterwards some checks ensure a valid distribution.

In the end we have an estimated hand for each player and we can differ between certain information (from step 1) and estimated information (from step 2 and 3).
In FreeDoko, the performance between virtual games with estimation and virtual games without estimation is nearly the same. So the performance benefit of the smaller hands of the estimation is nearly as good as the decreases because of the calculation.

Continuously improving the AI

Beside the previously discussed algorithmic approaches to the artificial intelligence, there is one more important issue in respect to the development of the artificial intelligence in FreeDoko.
As nobody is a perfect player and based on the great variety of rule combination it is impossible for us to check the artificial intelligence in all possible situations. Based on this early during the development of FreeDoko the mechanism of bug-reports where introduced. A bug-report is basically just a data collection to allow to reproduce a given situation. For this the bug-report contains the rule set of the current game, all played cards, including the used decision of the ai for this card and some relevant tournament information. By this it is possible to reproduce the exact situation for an ai decision by replaying card by card such a bug-report and updating accordingly the internal data of the ai and then to examine the decision of the ai in this situation and fix it if necessary.
For this part a special bug-report heuristic was implemented, which just plays the card from the bug-report.

As the artificial intelligence was improved with each version, it was soon also apparent that not all bug fixes were without effect to the rest of the ai and in some cases one bug fix for an ai-type in one game situation could create a problem in other situations.
The main reason for this is the strong dependencies between ai-types, heuristics and cards information, so that small changes in any part of the ai can have large side effects to the whole ai.
In order to test these side effects, we extended the bug-reports to a reference, where any action from the ai can be marked. The program checks whether the ai still decides according to the reference in the marked situation. So changes on the ai can be checked automatically for side effects.

Summary

This paper has shown the challenges in creating an artificial intelligence for the card game Doppelkopf and how those where solved in the program FreeDoko.
Special focus here was the usage of heuristics to find in many situations good cards without the need of time-consuming calculations.
But there is more one more important point in respect to heuristics. Heuristics can be

implemented in a way to give the artificial intelligence quite a human feeling when playing the game.

Because not the best but just a good a card is played many heuristics in FreeDoko just implement the “gut feeling” of a good Doppelkopf player. With this the artificial intelligence behaves much more humanly.
This is also one comment we got often from the users of FreeDoko. They mentioned how much they liked that FreeDoko just plays very similar to their real life card game buddies.
So the artificial intelligence in FreeDoko is definitely not perfect but plays as much as a human being as was possible for us to implement.

Anyhow who really wants to play against a perfect artificial intelligence he can nearly never beat?