Intelligent System for Playing Tarok

Mitja Luštrek
Jožef Stefan Institute
Jamova 39, 1000 Ljubljana, Slovenia
mitja.lustrek@ijs.si

Matjaž Gams
Jožef Stefan Institute
Jamova 39, 1000 Ljubljana, Slovenia
matjaz.gams@ijs.si

Abstract. *We present an advanced intelligent system for playing three-player tarok card game. The system is based on alpha-beta search with several enhancements such as fuzzy transposition table, which clusters strategically similar positions into generalised game states. Unknown distribution of other players' cards is addressed by Monte Carlo sampling. Experimental results show an additional reduction in size of expanded 9-ply game-tree by a factor of 184. Human players judge the resulting program to play tarok reasonably well.*

Keywords. Game playing, imperfect information, tarok, alpha-beta search, transposition table, Monte Carlo sampling.

1. Introduction

Computer game playing is a well-developed area of artificial intelligence. However, the majority of research has focused on games of perfect information, where players have complete knowledge of the entire game state. In chess and backgammon, computers can compete with top human players. For checkers, Othello and Scrabble, programs exist that can consistently defeat any human, while some simpler games, such as connect-four, have actually been solved (i.e. the best move for each possible position has been determined). The only major challenge left is probably the game of go.

On the other hand, games of imperfect information, such as card games where players typically do not see opponents' cards, have not enjoyed that much attention. An exception to this is bridge, where several programs exist and the best of them, like GIB [4], can challenge expert human players. Some work has also been done on poker, but even Poki [1], which is probably the only major effort in the field, is still mediocre compared to humans.

Tarok [7] is a very popular card game in several countries of central Europe. There are some tarok-playing programs [2, 9], but they do not seem to be particularly good and little is known of how they work. This is why we have chosen tarok for the subject of our research. The resulting program, Silicon Tarokist, is available online at http://tarok.bocosoft.com.

The paper is organised as follows. In section 2 rules of tarok and basic approaches to a tarok-playing program are presented. Section 3 gives the overview of Silicon Tarokist. Section 4 describes its game-tree search algorithm, i.e. an advanced version of alpha-beta search. In section 5 Monte Carlo sampling is described, which is used to deal with imperfect information. Section 6 discusses the performance of the search algorithm and presents the results of testing against human players. Section 7 concludes the paper and lists some ideas that might produce an even better tarok-playing program.

2. Playing Tarok

2.1. Rules of the Game

We have chosen three-player version of the game, which is used in tournament play (relatively unknown two-player and more complicated casual four-player versions also exist).

The game is played with 54 cards. There are eight cards of each suit and 22 trumps (called *taroks*). The black suits rank from lowest to highest: 7, 8, 9, 10 (1 point each; scoring will be explained at the end), Jack (2 points), Knight (3 points), Queen (4 points) and King (5 points). The red suits rank from lowest to highest: 4, 3, 2, 1, Jack, Knight, Queen and King (scoring is the same as for black suits). The lowest tarok (I) is called *pagat*, second highest (XXI) *mond* and highest (not numbered) *škis*[1] (5 points each and 1 point each for other taroks).

Each player is dealt 16 cards and the remaining six (called *talon*) are not yet revealed. Players then bid to decide who will be the

[1] This is the Slovene term; in German it is called sküs, in Hungarian skíz, in Czech škýz etc.

declarer. Each player in turn either names a contract or passes. Possible contracts from lowest to highest are *three* (three cards in hand will be exchanged for three cards in talon; 10 points), *two* (20 points), *one* (30 points) *solo* (no cards will be exchanged; 50 points), *beggar* (no tricks, i.e. cards played in one round, must be won; 70 points) and *klop* (when all players pass; the value of the tricks they win is subtracted from their score). Players are only allowed to name higher contracts than those before them and bidding ends when all players pass. Player with the highest contract becomes the declarer and the other two players oppose him jointly. Talon is revealed, the declarer chooses the appropriate number of cards from it, puts equal number of cards from his hand on his trick pile (where cards he wins are put) and the remaining cards are given to his opponents. The declarer may then make announcements: *kings* (20 points) means that he declares to win all four kings; *trula* (20 points) that he will win pagat, mond and škis; *pagat ultimo* (50 points) that he will win the last trick with pagat; and *valat* (500 points) that he will win all tricks.

The declarer plays the first card and the other players play theirs in turn. Suit must be followed if possible; tarok should be played otherwise. The trick is won by the highest card played of the suit led, unless it contains a tarok, in which case the highest tarok wins. In klop and beggar, higher card must be played if possible.

The value of the cards declarer has won is the sum of their point values, decreased by 1 per three cards. 35 is subtracted from it and if the remaining number is positive, the declarer has won the game. In any case this number is added to his score. If he has won, the point value of his contract is also added (and subtracted otherwise). Point value of fulfilled announcements is added as well (and those of unfulfilled subtracted). Scoring is cumulative over all the games in a session.

2.2. Tarok-playing Program

There are at least two basic approaches to computer tarok: previously prepared in-depth knowledge of the game can be employed and decisions can made based on basic knowledge of the game combined with extensive analysis of the situation at hand.

The advantage of the first approach is that complex and ingenious strategies, which human players have developed and tested, can be used. They only need to be expressed in a form that can be used by a game-playing program. However, applying such pre-made strategies to the variety of situations that arise in the course of a game is not easy. Unpredicted situations have to be dealt with. And of course care has to be taken that the program does not become predictable. Not to mention that humans are imperfect and so can be their strategies.

A variation of the above approach is to use machine-generated knowledge. Ideally game-theoretic optimal randomised strategy would be produced, but this is not possible due to enormous complexity of the task.

The more traditional approach is through search, which utilises computers' main advantages over humans: speed and perfect memory. In every position all moves are considered, for each move every countermove is tried etc. Such methods are completely adaptable and moves, being tailor-made for specific situations, are not predictable. However, even modern computers are sometimes not fast enough to execute such searches to sufficient depth.

We have decided on a combination of both approaches. The emphasis is on search, because we feel it has greater potential due to its flexibility. Here, the field of artificial intelligence offers a large array of methods to be used. Also, it is interesting to let computer devise strategies on its own and compare them to what human players do. Nevertheless, using human knowledge can be beneficial and sometimes even essential, so Silicon Tarokist employs it in several places.

3. Overview of Silicon Tarokist

Tarok-playing program must perform three distinct tasks: bidding, exchanging cards with talon and actual playing. When cards are dealt, it must determine how high it can afford to bid to be still able to win the game. If it wins the bidding, it must choose which cards from talon to take and which cards to remove from hand to achieve best hand. And lastly, it must decide which card to play each turn.

Despite these tasks seemingly being quite different, Silicon Tarokist performs them in basically the same way.

Bidding. For each contract starting with the lowest, a number of games are played out. The highest contract for which test games have been

won by a sufficient margin is considered viable. The program will bid as high as this contract.

Exchanging cards with talon. For each set of cards that can be taken from talon, several combinations of cards are considered for removal from hand. These combinations are such that if they are removed, program's hand contains no cards of one or more suits (since this is undoubtedly beneficial as tricks led by these suits will be won by playing taroks). For each combination, a number of games are played out and the best combination is chosen. The percentage of games where each possible announcement (kings, trula...) has been fulfilled is noted and if it is sufficiently high, the announcement is actually made.

Playing. The program uses Monte Carlo sampling: each time it needs to choose a card to play, it generates a number of random distributions of other players' cards. The game can then be treated as a game of perfect information. For each of these distributions, game-tree search is performed, which calculates the value of each card in program's hand. These values are then summed over all distributions and the one with the highest sum is played.

The margin by which test games have to be won during bidding is adaptable. When the program wins the bidding and looses the game, it must have been too optimistic, so the margin is increased. On the other hand, when it doesn't win the bidding and it wins the game anyway, it must have been too pessimistic, so the margin is decreased. Minimum percentages of fulfilled announcements during test games while exchanging cards with talon are adaptable in a similar fashion.

4. Game-tree Search

4.1. Alpha-beta Algorithm

Alpha-beta search [5] is the most widely used algorithm for playing games. Its purpose is to search game tree, which consists of nodes representing game states and edges representing moves. Values representing game score are assigned to leaves. The tree has interchanging max and min *plies*[2]: in the first it is your turn and in each node you choose the successor with the highest value (trying to maximize your score); in the second it is your opponent's turn and he chooses successors with lowest values. The algorithm employs (α, β) window, within which the value of the current node must be.

The algorithm used in tarok is slightly different because two players are teamed against one and so a third of plies are of one kind and two thirds of the other. Also, until three moves are made (and trick won), it is not known which kind of ply will follow, because it is not known which side will win the trick.

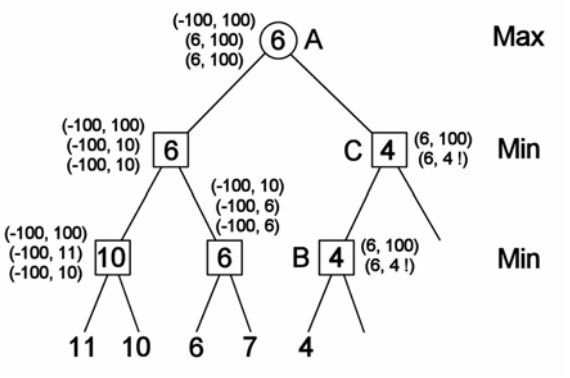

Figure 1. Alpha-beta search for tarok

Figure 1 shows alpha-beta search for tarok. First the left subtree of node A is expanded – it has value of 6. Then the right subtree is expanded, but in node B it becomes evident that the value will be no higher than 4, so in node A, left successor will be chosen. Thus the search can be cut short in nodes B and C. (α, β) windows can be seen next to each node.

Game-tree search usually cannot reach the end of the game because it would have taken too much time. Silicon Tarokist at first searches the tree to the depth of 9; after six cards have been played (and the branching factor of the tree decreases) the depth is 12 and after another three cards it is increased to 15. When the search is stopped, the nodes at the end have to be evaluated. Equation (1) shows part of the evaluation function (the rest deals with announcements and is not of particular interest).

$$f(\text{game\_state}) = \text{score\_difference} \times 5 + \\ + \text{taroks\_sum} + \text{mond} \times 50 + \text{škis} \times 50 \qquad (1)$$

Score_differece is the difference between program's and opponent's score. Taroks_sum is the sum of program's taroks and is needed to prevent taroks from being played too freely (which would otherwise happen, because they often win tricks, but should nevertheless be saved for opportunities where more valuable tricks can be won). Mond and škis indicate the presence of two cards in programs hand and are

[2] Ply is a term for tree level used in game playing, i.e. a single player's move or in tarok one third of a round.

needed to prevent them from being played unless the opportunity is particularly good (without this playing mond or škis often appears to be advisable, because not only are they likely to win tricks, but are themselves worth a lot).

4.2. Transposition Table

The same game state can often appear in multiple instances in a game tree – in fact, in tarok (and many other games), the flow of the game could be represented by a directed acyclic graph instead of a tree. Therefore it would be profitable to retain the values of expanded nodes and retrieve them when they are encountered again, so that they do not need to be more than once. Storing game states and their values is the purpose of transposition table [6]. Usually it is implemented as a hash table, which is also the case with Silicon Tarokist.

We have taken this idea a step further: we generalise game states by abstracting some strategically unimportant information. Thus only one game state from each set of equivalent game states has to be expanded and each time another game state from the same set is encountered, its value is read from the table. Similar approach has been used in GIB [4] with much success.

The following information is retained on our game states:

- the number of taroks;
- the approximate sum of taroks;
- the highest tarok;
- pagat, mond and škis;
- Jacks, Knights, Queens, Kings;
- the number of lower cards of each suit.

This results in 2.3 times less nodes being expanded (when searching to the depth of 9), while no impact on the quality of playing has been noticed.

4.3. History Heuristic

Game-tree search can be significantly sped-up if nodes are expanded in best-first order. If nodes that are 'too good' are encountered soon, alpha-beta algorithm ensures that searching is cut short. Figure 2 shows an example of this: in the lower tree nodes are ordered from best to worst, while in the upper tree the opposite is the case.

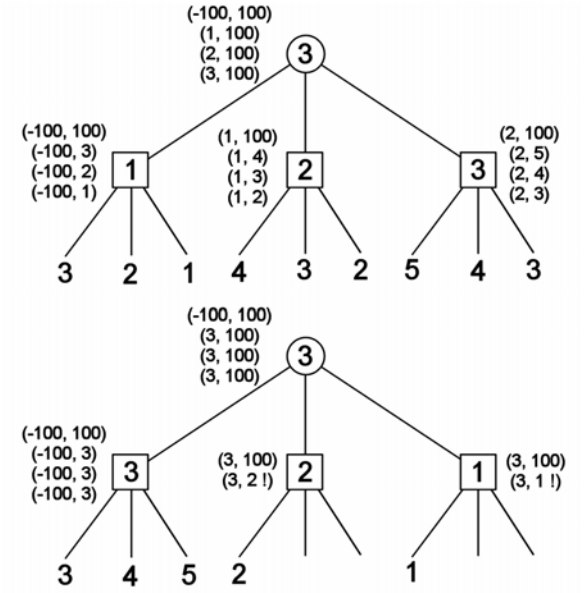

Figure 2. Move ordering

To order moves, a way of determining which moves are good without actually expanding them must be devised. History heuristic [8] assumes that plays that are good at one point will likely be good at another as well: if for example playing King of Hearts is good now, then it will probably be just as good after a trick of Spades has been played.

Each card is therefore assigned history value, which is initially 0 and is increased whenever that card causes a cutoff in alpha-beta search according to equation (2).

$$new\_value = old\_value + 2^{depth} \quad (2)$$

Depth is the depth of subtree under the node in which the card in question was played. The rationale for this equation is that if this subtree is deeper, it has been more firmly established whether the move in its root is good or not.

4.4. Minimal Window Search

Minimal window search [6] is an improvement upon alpha-beta based on the assumption that the first move considered at a given node is better then the remaining moves. First subtree is thus searched with the full window (α, β) and the remaining subtrees with the minimal window (v, v+1), where v is the value of the first subtree. If the remaining subtrees are indeed inferior, then the search is quicker due to smaller window (which causes more cutoffs). Otherwise additional work is required to do another search with the window (v+1, β). When move ordering is employed,

success is to be expected in sufficient number of cases for minimal windows search to be worthwhile. Silicon Tarokist uses move ordering and minimal windows search turns out to be beneficial.

4.5. Adjusting the Depth of Search

Some branches of game-tree are worth expanding more than others: an obviously bad move or a move that is very similar to another, but slightly worse, can be discarded. There are no general ways of doing this (apart from what alpha-beta algorithm itself does in this regard). Still, the potential profit is too significant to be ignored.

Silicon Tarokist uses human-made rules to select which moves will be considered. They are applied to the third card of each trick, because only after the first two cards have been played, there is enough information available for the rules not to be overly complex. All the rules cannot be listed due to space considerations, so we will only provide two examples:

- if tarok must be played, cards considered are: pagat, the lowest tarok, the lowest tarok to still win the trick and mond;
- if suit must be played and player on program's side has played higher card than the opponent, cards considered are: the highest card of the suit and the second highest card of the suit if the highest card of the suit is high enough to be able to win a trick later on.

5. Monte Carlo Sampling

In games of imperfect information, proven and efficient methods like alpha-beta search cannot be used, because it is not known which moves are available in min nodes (since other players' cards are unknown for example). But if a sufficient number of possible sets of other players' cards are generated, game-tree search can be performed on each of them and the overall best card can be played. Equation (3) shows how this is done.

$$f(\text{card}_i) = \sum_{j=1}^{n} P(\text{distribution}_j) \times \text{value}_{ij} \qquad (3)$$

$P(\text{distribution}_j)$ is the probability that j-th distribution of cards is the actual one. Value_{ij} is the value of i-th card for j-th distribution as computed by game-tree search. Since it is difficult to select representative distributions of cards, they are selected randomly and all probabilities equal 1/n. This is called Monte Carlo sampling and is used in Silicon Tarokist.

It can be shown that apart from its statistical nature, Monte Carlo sampling has several deficiencies [3]. Because it assumes that minimiser has perfect information, maximiser cannot take advantage of the fact that this is usually not the case. And if minimiser does in fact have perfect information, it can 'trick' maximiser into situations where the choice which would be right under most conditions is in fact wrong; this could in principle also be played around. Since it is generally impossible to determine how much information minimiser really has, these problems, although demonstrable, can usually be ignored. Somewhat more serious flaw is the fact that Monte Carlo sampling postpones decisions. For example, moves A and B are available. A wins if škis is held by the first opponent and the next move is C or if škis is held by the second opponent and the next move is D. Be wins regardless of who holds škis, except if all Hearts are held by the first opponent (which is unlikely). Monte Carlo will choose A, because it can win in all cases. But this is true only under the assumption that it will be known who holds škis by the next move. This being unlikely, B would be the better move. A method to address this has been developed, but it brings little benefit [4] and is not used in Silicon Tarokist.

A problem that has been observed is making bad moves, because they are wrongly considered inevitable. For example, playing King of Hearts, which in our sample is lost in most distributions of cards, and then playing King of Spades, which is won in most distributions, is considered equally good as first playing King of Spades and then King of Hearts. Therefore the first option is sometimes chosen. But if playing King of Hearts is postponed, it can sometimes be won after all. So it is prudent to make the more profitable moves first. This is effected by performing another game-tree search to the depth of one trick. The result of this search is then combined with the result of full search.

6. Experimental Results

To show the effect of enhancements of alpha-beta search, the algorithm has been run with several combination of enhancement. Each time the number of expanded nodes has been noted and the time spent for the whole search has been

measured. The time spent for each node has also been calculated. This has been done for the first card of the game and the depth of search 9 on a computer with Athlon XP P1800+ processor and 512 MB of memory. The results are shown in table 1. Nodes is the number of nodes expanded, t is the time used for the whole search and t/node is the time used to expand each node.

Table 1. Experimental results

Algorithm	Nodes	t (s)	t/node (µs)
all enhancements	8,667	0.248	28.6
no trans. table	25,349	0.527	20.8
no history heuristic	13,467	0.350	26.0
no minimal window	15,827	0.433	27.4
no adjusting depth	15,952	0.435	27.3
transposition table	95,575	2.451	25.6
history heuristic	324,633	5.463	16.8
minimal window	320,792	4.438	13.8
adjusting depth	258,060	4.891	18.9
no enhancements	1,598,924	21.266	13.3

All four enhancements combined cause 184-times less game-tree nodes to be expanded that in standard alpha-beta search, for which 84-times less time is needed (because the enhancements increase the time spent per node). Transposition table has greatest impact, even though its use is most time-consuming.

Human players judge Silicon Tarokist to be good enough for playing against it to be interesting. It does make occasional mistakes and sometimes seems to be playing 'aimlessly'. This can probably be attributed to its search not being deep enough, so long-term strategies cannot be developed. On the other hand, sometimes it plays in ways that could be called cunning if it were human. For example, the players before the program have played low cards. The program hasn't won the trick with King, although it had it, but rather with Knight, saving King for the next turn when it won Queen with it.

7. Conclusion

Silicon Tarokist can manage all aspects of three-player tarok and can play it reasonably well. Its game-tree search algorithm is a significant improvement upon alpha-beta, but in itself is not enough for high-level play. It is sufficient towards the end of the game, but quite inadequate at the beginning.

Polishing the game-tree search is left for future work and some variant of best-first search (perhaps proof-number search) should be tried. But we feel that this will not be enough: another way will have to be found for the program to develop long-term strategies.

Of the enhancements we have used, our version of transposition table brings the greatest benefit (which it would not if it were not augmented by knowledge of tarok) and the one which has most room for improvement is adjusting the depth of search, which is entirely knowledge-based. So adding knowledge to our program seems to be the most promising way to further improve it.

8. References

[1] Billings D., Davidson A., Schaeffer J., Szafron J. The Challenge of Poker. Artificial Intelligence 2001.

[2] Cigan S. Tarok. http://www.tarok.net [02/25/2003]

[3] Frank I., Basin D. A Theoretical and Empirical Investigation of Search in Imperfect Information Games. Theoretical Computer Science 1999.

[4] Ginsberg M. L. GIB: Imperfect Information in Computationally Challenging Game. Journal of Artificial Intelligence Research 2001; 14.

[5] Ingargiola G. P. MiniMax with Alpha-Beta Cutoff. UGAI Workshop; 1996. http://yoda.cis.temple.edu:8080/UGAIWW W/lectures96/search/minimax/alpha-beta.html [02/25/2003]

[6] Marsland T. A. A Review of Game-Tree Pruning. Journal of Computer Chess Association 1986; 9(1).

[7] Obradovic Z. Slovenian Tarok. Card Games. http://www.pagat.com/tarot/sltarok.html [02/25/2003]

[8] Schaeffer J. The History Heuristic and Alpha-Beta Search Enhancements in Practice. IEEE Transactions on Pattern Analysis and Machine Intelligence 1989.

[9] World Association for Tarok. http://members.lycos.co.uk/tarok [02/25/2003]