Bachelor Mathematics

Bachelor thesis

Analysis of game Hanamikoji

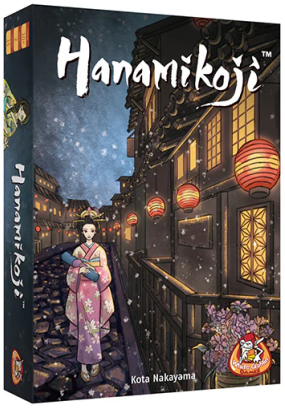
by

Daniela Mackay

July 1, 2024

Supervisor: Mark van den Bergh

Second examiner: Ger Koole



Department of Mathematics
Faculty of Science

VU VRIJE UNIVERSITEIT AMSTERDAM

Abstract

We apply a Monte Carlo Tree Search (MCTS) algorithm to identify optimal strategies in the two-player game Hanamikoji. By simulating numerous matches between the MCTS agent and a random player, we analyse the decision-making patterns of the algorithm to discern effective strategies.

Title: Analysis of game Hanamikoji
Author: Daniela Mackay, d.m.mackayparrott@student.vu.nl, 2663452
Supervisor: Mark van den Bergh
Second examiner: Ger Koole
Date: July 1, 2024

Department of Mathematics
Vrije Universiteit Amsterdam
de Boelelaan 1081, 1081 HV Amsterdam
http://www.math.vu.nl/

Contents

1. Introduction

The analysis of games has for the past 75 years been a relevant field in artificial intelligence and mathematics. The first research recorded to be in 1948 [4, 3], where the game of Nim was analysed and a computer opponent was constructed based on this theory. The form of decision making during games offers a practical platform to test theories and yield clear, interpretable results. The field of game theory commenced by looking at open information deterministic two player games, that is games where all information is known to both players (as opposed to having hidden cards) and where there is no element of chance (deterministic). With the help of machine learning we can take one step further and look at games in which all information is not known and rely on chance.

The game we have chosen to analyze is Hanamikoji, a two player game of imperfect information. During the game each player has 4 turns. In each turn they must take one out of four actions, which can only be used once. Two of these actions are to offer trades. Many factors have to be considered such as the order in which to use the actions, and how to combine cards optimally during trades. Therefore, each decision carries a lot of weight.

This is where we would like machine learning to help us. The tool we have chosen to use is Monte Carlo Tree Search, a reinforcement learning algorithm that takes random paths down a decision tree until reaching the final outcome, the win or loss of the game, and then walks backwards updating all nodes on the outcome.

This algorithm is commonly used in the analysis of games, as the decision tree structure is the most convenient representation. Some examples of research conducted on other games include The Crew [2] and Dou Di Zou [5].

2. The Game

The game Hanamikoji is a two-player card game in which one aims to win the favour of geisha's. On the table lay seven geishas: 3 twos, 2 threes, 1 four, and 1 five. These numbers (2,3,4,5) represent how many cards of each colour there are in the deck, totalling 21 cards.

Figure 2.1. Image of cards

Each player starts with six cards and draws one card at the beginning of each turn. At the end, there is one card remaining in the deck. During the game, each player takes turns doing actions. The four actions are:

- Store: put aside one card to be played last.
- Discard: put aside two cards never to be played.
- Trade 3x1: offer three cards, the opponent chooses one to play immediately on their side, and the remaining two are played on the current player's side.
- Trade 2x2: offer two pairs, opponent chooses one pair to play immediately on their side, and the remaining pair is played on the current player's side.

Each of these actions can only be used once per round. This amounts to six decisions per player every round, four being which action and cards to play each turn, and two are which cards to pick from a trade.

Playing a card means placing it on your end of the corresponding geisha. There are two objectives in the game, either gaining the favour of four geisha's or the sum of the numbers of the geisha's whose favour we have to add up to 11 or more. It is possible for both players to achieve an objective; in that case, the player with 11 or more points wins. To win the favour of a geisha, one must have the majority of cards played on your side of it. If neither objective has been met, then a new round is started, where the favours won are remembered by tokens. During the second round, the tokens will act as a tiebreaker in the case that a geisha has equal amount of cards for both players and give the favour over to the player with the token on that card.

2.1. Probabilistic breakdown

We are interested in building some theoretical background to analyse the decisions. In game theory, the bottom up approach is often used. It consists of looking at the last step of a game to be able to discretely analyse the consequences of one move.

Let us take a look at the last action of a game, where Player 2 only has the 2x2 Trade remaining. Figure 2.2 illustrates how cards are distributed before the trade.

Figure 2.2. Visualization of table during the last turn.

This is the easiest to compute because it is when we have the most information and least possible actions. We have four cards in our hand and all other cards are known except for four: the last card in the deck, one card the opponent has stored, and two he has discarded. Out of these four unknown cards the only relevant one is which card has been stored as this will be played later. Therefore, we should be able to calculate which pairing combination to offer in order to have the highest chances of winning. We do this by calculating the outcome of offering a trade, assuming that we know which card is stored.

More specifically, say we have cards a b c d, which have possible pairing combinations A = (ab) (cd), B = (ac) (bd), and C = (ad) (bc) (less if there are repeated cards). The unknown cards are $U1$, $U2$, $U3$, $U4$. For each of these combinations, there are two branches: that the opposing player chooses trade 1 or 2.

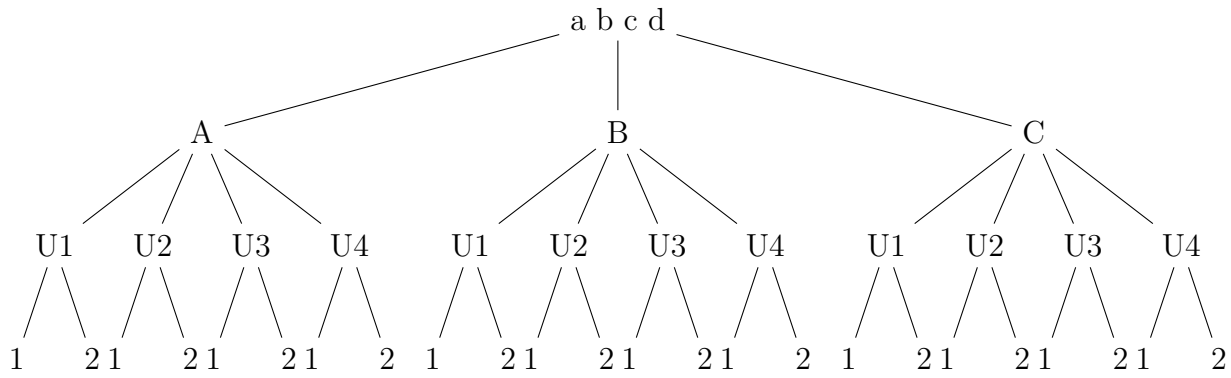


Figure 2.3. All possible outcomes of the last turn of the round.

Applying this to our example results in Figure 2.4 where 1 is a win and 0 is a loss or tie, and each level shows the average of the outcomes.

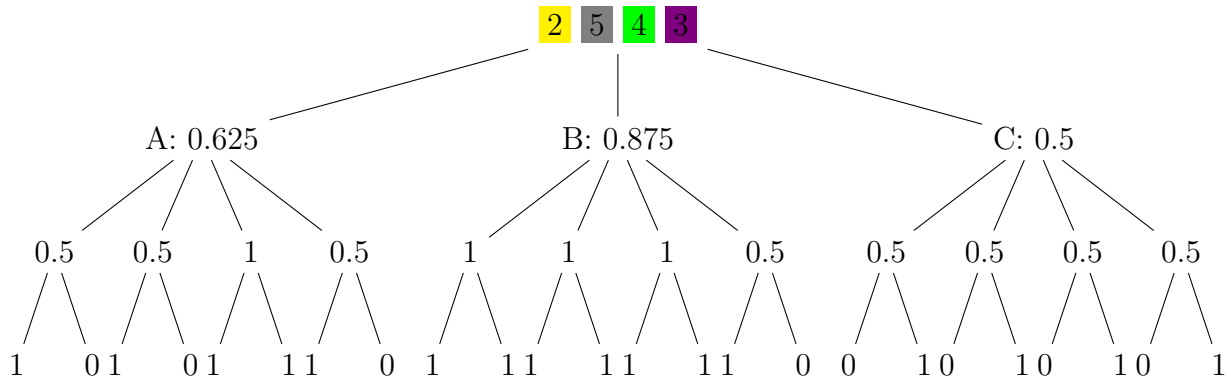


Figure 2.4. Decision tree applied to a real game. With U1: 5 U2: 5 U3: 3 U4: 2 .

3. Monte Carlo Tree Search

3.1. The algorithm

The Monte Carlo Tree Search (MCTS) algorithm is based on random walks through a decision tree, which makes it ideal for applying to a card game. It consists of four steps, illustrated in Figure 3.1.

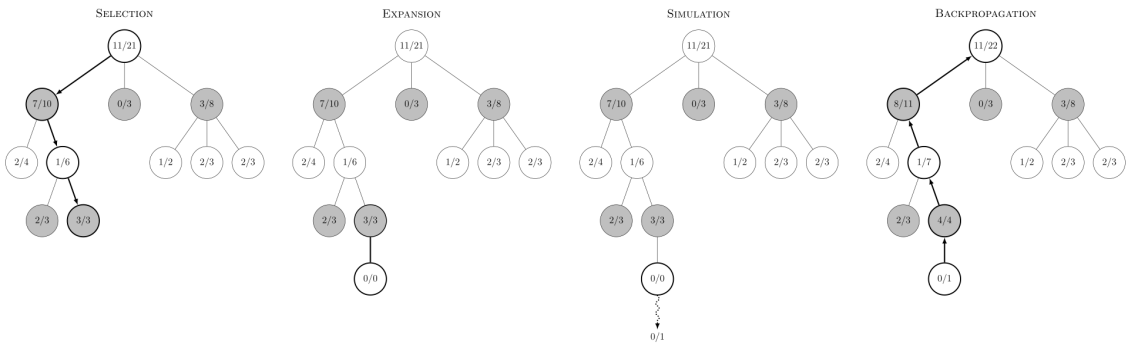


Figure 3.1. Diagram of Monte Carlo Tree Search algorithm

- Selection: find a node that isn't fully expanded.
- Expansion: take an untried action resulting in a new child node.
- Simulation: randomly playout the game until the end, returning 1 for a win and -1 for a loss.
- Backpropagation: update fraction $\frac{\text{wins}}{\text{times visited}}$ in current node and all parent nodes.

We thus build a probability function of how likely one is to win if this path is chosen. Once back at the parent node, we repeat the process with another randomly chosen child node. The algorithm is based on the Law of Large Numbers, where by simulating n random walks we create an unbiased sample group on which we can build our probability function on. As always, the larger the amount of simulations, the more accurate the outcome will be.

In order to balance computation time and the quality of the results, the number of simulations was tailored according to the number of possible actions as can be seen in Table 3.1.

Number of possible actions	Number of simulations
> 50	1000
$50 > x > 4$	300
$4 \geq x > 1$	100
1	1

Table 3.1. Number of simulations depending on possible actions.

There is a function in the code which is worth mentioning: the best child function. It is applied during two steps: during the selection step, when all child nodes have been visited, we will have to revisit a node and expand it one step further. To choose which node to revisit we use the output of the best child. The other time it is applied is after all simulations are run, in order to output which action it recommends taking.

The function applies the Upper Confidence Bound (3.1) to all child nodes of the root node.

$$\frac{\text{wins}}{\text{times visited}} + c * \sqrt{\frac{\text{number of simulations}}{\text{times visited}}} \tag{3.1}$$

This formula balances exploitation versus exploration. The first term corresponds to exploitation, when this is larger the algorithm tends to repeat paths that have had the best performance in the past. The second term corresponds to exploration: it makes sure we don't stick a good path early on and encourages the expansion of new nodes. The parameter c adjusts the balance between both terms. During selection we use $c = 1.4$ and for the final output $c = 0.2$, since exploration is no longer a priority.

3.2. Set up

We conducted 150 trials of a Monte Carlo agent versus a random player. For every decision that the Monte Carlo agent has to take, either taking an action during their turn or choosing which trade to accept from their opponent, a new decision tree is formed with the current state at it's root. From here the Monte Carlo Tree Search algorithm is applied outputting the best move which will be played on the trial.

From each of these trials we collected three sets of data: which moves where executed during the trial games, the winner of the trial, and a file containing the output vector of the best child function. That is, a vector in which the Upper Confidence Bound formula has been applied to all child nodes after all simulations have been finished. This is interesting because there there are often ties where the maximum of the vector are equal, so we grouped all of these child nodes to look into which choices were most favourable.

For the implementation we followed the template from ai-boson (MIT) [1]. For this, a simulator of the game had to be built in order to represent each state and to be able to move from one state to another.

4. Results

The win rate of all simulations amounted to 124 from the Monte Carlo agent and 26 from the random player.

Table 4.1 shows the order of moves that were taken during each trial.

	First Move	Second Move	Third Move	Fourth Move
Discard	17	41	25	69
Store	5	22	80	45
Trade 2x2	88	41	17	6
Trade 3x1	42	48	30	32

Table 4.1. Counts of which actions have been taken by the Monte Carlo agent in for each move

Here we can see that, for the first move, there is a clear preference to use the Trade 2x2. The preferred order of playing cards would be Trade 2x2, Trade 3x1, Store, and Discard.

Therefore, we would like to look deeper into the Trade 2x2. We created some categories to classify these pairs in, they are:

1. Low (or High) pair: one of the pairs are two of the same cards.
2. Triple : three of the same cards within the two pairs
3. Two high vs two low: for example, 4 & 5 vs 2 & 3.
4. Highlow Highlow: for example, 3 & 5 vs 3 & 5

We analyse the pairs offered in the Trade 2x2 from the trial moves and the best children from all the simulations to find the pattern count in Table 4.2.

Here we see that the two most successful combinations are two low vs two high, and Highlow Higlow. In particular, Highlow Highlow is the most used strategy with about double as many appearances as the rest. This is very interesting since this was a naturally reoccurring pattern that we found while playing in real life against many opponents.

	Actual Moves	Simulation
Low pair	26	964
High pair	10	835
Triple	4	147
Two low, vs two high	5	1487
Highlow Highlow	47	3276

Table 4.2. Counts of patterns found in Trade 2x2.

5. Conclusion

Our analysis has revealed compelling insights into the strategies and patterns inherent in the game of Hanamikoji. By first taking a theoretical standpoint and then through the application of Monte Carlo Tree Search, we explored decision-making strategies and were able to navigate the complexities of imperfect information and probabilistic outcomes.

Further work

A deeper investigation into the deterministic perspective of the game analysis, using probability and game theory to breakdown the possible last moves and the chances of winning given having one hand. In addition, it would be interesting to modify the Monte Carlo agent with the theory to be able to make more educated decisions.

Bibliography

[1] Ai Boson. Monte carlo tree search (mcts) algorithm. `https://github.com/ai-boson/mcts`, 2022.

[2] Aron de Jong. An analysis of the cooperative card game the crew. Bachelor's thesis, Universiteit Leiden, 2021.

[3] Ferranti Ltd. *Faster Than Thought. The Ferranti Nimrod Digital Computer. A Brief Survey of the Field of Digital Computing with Specific Reference to the Ferranti Nimrod Computer*. 1951.

[4] Raymond Redheffer. A machine for playing the game nim. *The American Mathematical Monthly*, 55(6):343–349, 1948.

[5] Daniel Whitehouse, Edward Powley, and Peter Cowling. Determinization and information set monte carlo tree search for the card game dou di zhu. pages 87 – 94, 10 2011.

A. Python Implementation

- Link to Github