Riichi Mahjong

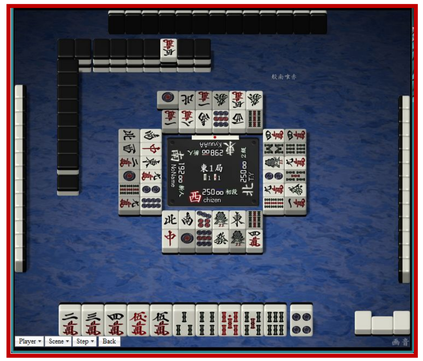
Elliot Miller, James WoMa, Nantanick Tantivasadakarn

Stanford University

LELAND STANFORD JUNIOR UNIVERSITY
DIE LUFT DER FREIHEIT WEHT
1891

Building Hands

Riichi Mahjong is a Japanese variation of a tile-based game developed in China that is similar to Gin Rummy. Players draw tiles from a wall or from opponent's discards in order to form a complete hand out of smaller melds. Mahjong players must be able to quickly determine best tiles to discard, factoring in the probabilistic nature of draws and the danger presented by opponent's hands.



We built AI for an online mahjong platform, Tenhou.net, using several approaches to evaluating game state and planning subsequent moves.

Due to the lack of training data and the fact that each round takes 11-30 minutes, our AI will be focussing on win rate and will not be directly evaluating opponents hands. Specifically, we will explore using shanten as a heuristic to define utility for our AI.

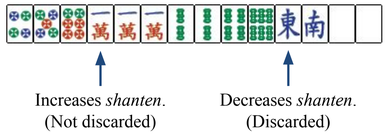
Features

Features available to the AI included the entire visible game state, including discarded tiles for each player, tiles revealed by opponents, and tiles in hand.

Given a hand, we can calculate a statistic called *shanten*, or the minimum number of tile draws required to reach *tenpai*, a hand one away from winning. Assuming that all tiles are available, a lower *shanten* indicates a strictly better hand.

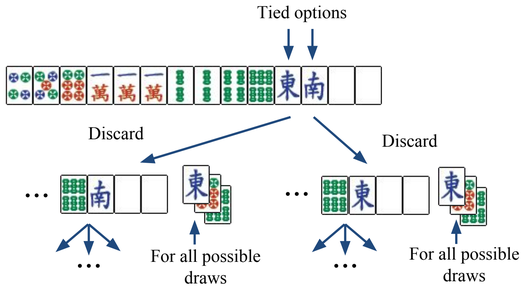
Models

1. Greedy *Shanten* Optimization



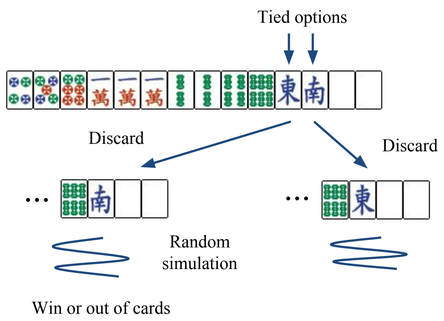
This method greadily selects a tile that decreases shanten. If two tiles are tied, it will selects the first tile.

2. Search-Based *Shanten* Algorithm



This second method is similar to the first but does a tie breaker by searching all possible draws and recursively calculate the expected *shanten* on each discard.

3. *Shanten* Monte Carlo Search



This method replaces searching all possibilities in the second algorithm with calculating many simulations of a randomly playing agent. The utility is the sum of *shanten* on each simulation.

4. Monte Carlo Simulation

This method looks at all discard options and calculates the simulated average squared sum *shanten* on each option.

Results

After running multiple games on Tenhou.net, we have calculated the average hand win rate, feed rate (providing a win tile to an opponent), round win rate, and average rank after a round. Statistics were also calculated without *ryuukyouko*, or when a game ended in the tile wall being exhausted and no player completing a hand.

The best algorithm is the search based algorithm. However, all algorithms are worse than a previous rule based algorithm (Nihisel's AI).

Algorithm → / Metric ↓	Greedy *Shanten* Optimization (n=678, r=122)	Search-based *Shanten* Optimization (depth = 3, n=899, r=156)	*Shanten* MCS (n=739, r=121)	Monte Carlo Simulation (n=1088, r=193)	Nihisel's AI (provided bot, n= unknown, r=600)
Hand Win Rate (excluding *ryuukyouko*)	0.1077 (0.1189)	0.1835 (0.2010)	0.1407 (0.1583)	0.1489 (0.1667)	0.1997
Feed Rate (excluding *ryuukyouko*)	0.2286 (0.2524)	0.1913 (0.2095)	0.2016 (0.2268)	0.2242 (0.2510)	0.1088
Round Win Rate	0.0328	0.0956	0.0909	0.0933	0.2241
Average Rank	3.37	2.87	3.10	3.03	2.53

Discussion

The search-based *shanten* optimization got very close to an average 0.25 hand win rate. MCS and Monte Carlo simulation performed worse despite being more powerful and not limited by depth. This is likely since the search algorithm provides more consistent heuristics for breaking ties. Feed rates were high across the board since all AI aggressively tried to complete hands with no regard for opponents, and any differences are likely due to better hand formation.

A fault in the AI was they they were unable to consider *yaku*, or point value of a hand necessary to call a win. Thus, the AI preferred winning many low-value hands instead of holding out for large wins, resulting in low round win rates and rank. This can be seen in comparison to the AI provided with the bot, which was based on professional mahjong strategies rather than AI models. In addition, an unknown bug prevented the bot from calling *riichi*, an announcement of being one tile from winning that serves as a common win condition, likely resulting in reduced win rates.

Future Work

As of now, the AI considers opponent's discards as one agent. Separating opponents and analyzing their actions would allow for better defensive play.

Obtaining a database of mahjong games would allow us to better tune our feature evaluations and build self-training algorithms.

References

[1] Nihisel, Alexey, Mahjong Repository, tenhou-python-bot. GitHub, 2018.
[2] Nihisel, Alexey, Mahjong Repository, mahjong. GitHub, 2018.
[3] European Mahjong Association, Riichi Rules for Japanese Mahjong, 2016.
[4] Loh, Wan Jing, AI Mahjong. CS229, 2009.