Mahjong AI GuoShiWuShuang

Chaozhe Kong

Peking University

January 6th, 2021

Overview

GuoShiWuShuang is a Chinese Standard Mahjong AI focused on the tile efficiency.

On the tile efficiency, GuoShiWuShuang uses two different algorithms separately for hand tiles with big/small shanten.

GuoShiWuShuang also tried to consider defense, but it failed to achieve a good effect.

Tile Efficiency

Tile efficiency is a process of hand development, where a player seeks to make maximized use of either draws and/or discards to attain or even complete the hand as fast as possible.

In the tile efficiency, we pay little attention to other players' behaviors. It is just like a single-player game.

For convenience, just the algorithms for discarding are introduced below. That is, which tile should the player chooses to discard in his own round. The algorithm for deciding whether Pong/Kong/Chow is approximate.

Tile Efficiency

Shanten is the number of tiles needed for reaching tenpai. In other words, a player needs at least shanten+1 rounds to win without considering the limit of points. (In Chinese Standard Mahjong, winning tiles must have points greater than 8) When shanten is small, we will have enough time to search out all possible situations to win in the least rounds. Otherwise, there exists two greedy algorithms.

Search Algorithm

For convenience, we simplify the game as a single-player game with only drawing and discarding. In each round, the only player draws a tile from all the leaving tiles with uniform random and chooses one of his tiles to discard.

Define $P(S, l)$ as the maximal probability that the player with tiles set S can win in just l rounds. Here S is a multiset.

Suppose the least rounds we need to win is l, then we can just discard the tile t with maximal $P(S \setminus \{t\}, l)$.

Search Algorithm

Define $win(S) = 1$ when the player with tiles set S has won, and $win(S) = 0$ otherwise.

We have the following expression to calculate $P(S, l)$:

$$P(S, l) = \frac{\sum_{i=0}^{|T|-1} P'(S \bigcup \{T_i\}, l)}{|T|}$$

Where $T_0, T_1, \ldots T_{|T|-1}$ are all leaving tiles,and

$$P'(S, l) = \begin{cases} win(S) & l = 0 \\ \max\{P(S \setminus \{t\}, l-1) | t \in S\} & l > 0 \end{cases}$$

Some optimizations are used in my implementation to make it faster.

Fast Greedy Algorithm

Enumerate the tile to discard and use dynamic programming to calculate the minimal number of rounds to win and the probability of the leaving tiles approximately.

For convenience, we simplify the game as a single-player game same as before and only consider the winning form of 4 melds and 1 pair.

The dynamic programming includes two parts:

(1) calculate the minimal round to get i melds and j pairs $(i = 0, 1, 2, 3, 4; j = 0, 1)$ and the approximate probability for Dot, Bamboo, Character and Honor separately.

(2) merge the output of (1).

Fast Greedy Algorithm Part1

This problem is trivial for Honor and same for Dot, Bamboo and Character. So we only consider the approach for Dot.

Define $minRound(i, meldCount, pairCount, c_1, c_2)$ as the minRound to get $meldCount$ melds and $pairCount$ pairs using Dot $1, 2 \dots i$, and there are c_1 Dot $i-1$ and i left for get melds of Dot $i-1, i, i+1$ in future, and c_2 Dot i left for get melds of Dot $i, i+1, i+2$ in future.

The result we want is $minRound(9, meldCount, pairCount, 0, 0)$.

It is easy to transfer and calculate the approximate probability at the same time.

Fast Greedy Algorithm Part2

Think Dot, Bamboo, Character and Honor as different items, this problem is similar to the knapsack problem.

Define $minRound(i, meldCount, pairCount)$ as the minRound to get $meldCount$ melds and $pairCount$ pairs using item $1, 2, \ldots i$.

The result we want is $minRound(4, 4, 1)$.

It is easy to transfer and calculate the approximate probability at the same time.

Slow Greedy Algorithm

The disadvantage of the previous algorithm is that it can't consider the limit of points during dynamic programming, because the rule of point calculating is very complicated.

I observe that, in most cases, the number of possible final winning tiles with the least rounds from current tiles is small. So it allows us to search out all possible final winning tiles, calculate their points and approximate contribution to the probability of each tile.

This Slow Greedy Algorithm may exceed the time limit in a few cases. In this situation, we change to use the Fast Greedy Algorithm.

Thanks for listening!