Mahjong AI GuoShiWuShuang

Chao-Zhe Kong
Peking University
kcz@pku.edu.cn

Abstract

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

1 Introduction

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.

2 Tile Efficiency

The tile efficiency[1] 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 algorithm 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.

Shanten [2] is the number of tiles needed for reaching tenpai. In other words, a player needs at least shanten+1 rounds to without considering the limit of points. When shanten is small, we will have enough time to search out all possible situations to win in the least rounds. Otherwise, there exists a efficient greedy algorithm.

2.1 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 possibility 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 $P(S \setminus \{t\}, l)$.

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

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

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

2.2 Greedy Algorithm

Enumerate the tile to discard and use dynamic programming to calculate the minimal round to win and the possibility of the leaving tiles approximately.

For convenience, we simplify the game as a single-player game same as before and only consider the win 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 possibility approximately for Dot, Bamboo, Charater and Honor separately.

(2) merge the output of (1).

part 1

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

Define $minRound(i, meldCount, pairCount, c1, c2)$ as the minRound to get $meldCount$ melds and $pairCount$ pairs using Dot $1, 2 \ldots i$, and there are $c1$ 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.

It is easy to transfer and calculate the possibility approximately at the same time.

part 2

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

[1] http://arcturus.su/wiki/Tile_efficiency

[2] http://arcturus.su/wiki/Shanten

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

It is easy to transfer and calculate the possibility approximately at the same time.