Chinese Standard Mahjong AI Based on the Searching Algorithms with Traditional Evaluation Functions

Yian Wang[1] , **Kunhang Lyu**[1] and **Zhehuan Chen**[1]
[1]Peking University
{1900012980, kylelv,1900013017}@pku.edu.cn

Abstract

The game of Mahjong has a long history of more than 2,000 years, and many researches have been done on this charming and challenging game to explore its origin. During this competition, we create a Mahjong bot based on searching algorithm with traditional evaluation functions. On this foundation, we implement some optimizations to improve the accuracy and efficiency. This paper introduces the details of our work.

1 Introduction

As a popular competitive Chinese entertainment in the world, Mahjong has a history of more than 2,000 years [1]. A number of theories have been presented to explore its origin of probability and game theory. It's because of its interest and challenge on thinking that educators usually apply it to courses as an excellent example. Our research, which is the improvement of our course project, was done on "Chinese Standard Mahjong" – one of the commonest rules of Mahjong.

2 Glossary List

To begin with, the glossary list is shown to give a clear account of some concepts mentioned in the following descriptions of our research.

Tiles: the basic units of Mahjong with specific contents on them. They are Character suit(1w-9w), Bamboo suit(1s-9s), Dot suit(1p-9p), Wind tiles: East(1z), South(2z), West(3z), North(4z) and Cardinal tiles: Red Dragon(5z), Green Dragon(6z), White Dragon(7z). A tile is presented as lower-case letters (such as x and y).

Patterns: a set of tiles each player holds. A Pattern is presented as capital letters (such as S and T).

Chow: an operation of claiming a discarded tile from the player on the left hand to form a sequence, presented as $chow(x)$.

Pong: an operation of claiming a discarded tile from the other players to form a triplet, presented as $pong(x)$.

Kong: an operation of claiming a discarded tile from the other players to form a kong, presented as $kong(x)$.

Hu: victory in one game of Mahjong. The winner is the first to hold tiles consisting of four groups of triplets, sequences or kongs and a pair of tiles meanwhile whose total score of his pattern is not less than 8. There are many specific winning patterns.

Claiming tiles: operations of Chow, Pong and Kong. A claiming operation is presented as $op(x)$.

Winning hand: the winning pattern.

Winning structure: $ST(S)$, details of a winning hand S, including tags of each group in the winning hand presenting whether the group is formed by drawing or claiming.

Points: total scores of a winning pattern.

Fan: specific winning patterns with corresponding scores. Points will be added if a winning hand matches a "Fan". A winning hand can match several kinds of "Fan".

Least number of tiles to a winning hand: $LWT(S,T)$ presents the least number of tiles needed to be changed to achieve a winning hand y from the current pattern x.

Least number of tiles to Hu: $LHT(S) = min\{LHT(S,T)\}$. It describes the distance between the current state and the winning state – the less, the easier to get victory.

Winning plans: a set of winning hands $P(S)$. $P(S) = \{S|LWT(S,T) = LHT(S)\}$.

A useful tile: a tile u considered useful if $LHT(S+u) < LHT(S)$.

Addition/subtraction between a pattern and a tile: $S \pm x$. Assuming that $S = \{x_1, \cdots, x_n\}$ is a pattern and x is a tile, we have $S + x = S \cup \{x\}, S - x = S - \{x\}$.

Addition between a pattern and a claiming operation: $S + op(x)$. Assuming that $S = \{x_1, \ldots, x_n\}$ is a pattern and $op(x)$ is a claiming operation, we have $S + op(x) = S'$ and S' satisfying $S' = S \cup \{x\}$ and the corresponding group in $ST(S')$ is relabeled.

3 Basic Work

3.1 Evaluation Function

The game of Mahjong is essentially a dynamic game of incomplete information with a large number of states. To solve

this kind of problem, searching is one of the commonest methods. However, as the numerous states can never be completely accessed in quite limited times, an effective evaluation function is required to be carefully designed. That's $E(S)$ we use in our program. It's calculated using $P(S)$ and $LHT(S)$, and with the assistance of the evaluation function, many decisions are easy to make. For instance:

1. Assessing the weight coefficients of expectations of the current patterns winning the game. For the current pattern S, calculate the probability of achieving each pattern in $P(S)$.

   Firstly, to simplify the model, we assume that all tiles will be drawn at a same possibility of p, and claimed at a possibility of $3p$ (when Ponging and Konging because of discarding of the other three players) or p (when Chowing from the player on left hand).

   Secondly, we count the remaining (not explicitly discarded) number of each useful tile x as $r(x)$ then multiply it to the result.

   Thirdly, consider that we can claim tiles only when we have already held two tiles in a group, therefore possibility of claiming a useful tile is actually lower than we calculate above. We define some suitable parameters presenting the possibility of forming a group by Ponging or Chowing when lacking 2 or 3 tiles (respectively as $p_{p2}, p_{p3}, p_{c2}, p_{c3}$, and satisfying $p_{p2} < (3p)^2, p_{p3} < (3p)^3, p_{c2} < p^2, p_{c3} < p^3$).

   Fourthly, we give another reduction to the possibility weight of Konging by reducing every tile's weight to one-third of the original as it contributes little to the result.

   In addition, we take the special Fan "seven pairs" into consideration. Complexity of combinations influences that the average possibility of reaching a winning hand decreases, so we redefine the possibility of drawing a tile as $p_7(p_7 < p)$. Last, make a product of the possibility of all useful tiles for every winning pattern and sum them over. The output $E(S)$ is used as the expectation's weight coefficient of the current pattern.

2. Assessing the importance of each tile when needed to discard a tile. For every tile x enumerated in hand, calculate $E(S-x)$. The larger $E(S-x)$ is, the less important x for the current state is. Obviously, we choose x satisfying $E(S-x) = min_{x \in S}\{E(S-x)\}$as the discard in this turn.

3. Assessing the operations of Chow, Pong, Kong at the moment. For the claiming operation $op(x)$, calculate $E(S+op(x))$ and judge whether $E(S+op(x))$ is greater than $E(S)$ or not. If so, accept the claim, ignore otherwise.

Actually, this method of assessment is reasonably in accord with intuition. Once we implement a claim operation, the number of winning plans probably decreases. For example, assume that we hold a short pattern "2s 2s 3s" then comes the tile "2s". If we choose to implement a Pong operation, we will lose the probability of forming the "1s 2s 3s" or "2s 3s 4s" group. The possible drawing of "1s" in the following turns can be quite embarrassing. It reflects in our evaluation function that $E(S)$ is positively correlated to $|P(S)|$ while negatively correlated to $LHT(S)$. Although claiming operations will reduce $LHT(S)$, $|P(S)|$ can decrease as well. As a result, if the effect of the latter is great enough, the claiming operation will bring negative returns. This analysis is usually ignored by Mahjong beginners.

3.2 LHT and Winning Plans

The major work of the program is to generate all winning plans $P(S)$ for the pattern S we hold every moment. It takes our program majority of executing times. The searching begins with the groups formed by claiming operations in the current pattern as the initial state. Then we apply functions $Search\_dfs\_triplet(dep, lim, I, J)$ to search the triplets then $Search\_dfs\_sequence(dep, lim, I, J)$ to search the sequence in order to produce a winning pattern. The greedy idea of this searching's order assists improving efficiency and accuracy. Once reaching a possible winning pattern, we use the function $judgeHU()$ to judge whether the points of this pattern is not less than 8. If so, we successfully get a winning hand T. Then $LWT(S,T)$ is easy to calculate (actually calculated dynamically in the searching processes), and $LHT(S)$ and $P(S)$ are updated. The total number of state has a theoretical upper bound of $4 * 52^5 \approx 1.5 \times 10^9$, which is quite unacceptable because the function $judgeHU()$ runs as slowly as crawling. We hope to control the state's number under a level of 10^5, which is what we mainly do in our program introduced in the following.

Pseudo codes of the mentioned functions are shown:

```
void search_best(int dist){
for (i,j) in pair_available_set
  if !isok_pair(i,j) then
   continue
  end if
  TMP.add(pair(i,j));
  dist <- dist+getdiff()
  if(judgeHU14()) then
    plan.add(TMP);
    update(leastHuCard);
  end if
  recover(TMP);
end for
}
void search_dfs_shun(int dep,int limit,\
int I,int J){
  if dep==0 then
    search_best(diffCardlimit-limit);
    return ;
  end if
  for (i,j) in shun_available_set
    if !isok_shun(i,j) then
     continue
    end if
    TMP <- TMP+shun(i,j);
    diff <- getdiff();
    search_dfs_shun(dep-1,\
```

```
    limit-diff,i,j);
    recover(TMP)
  end for
}
void search_dfs_ke(int dep,int limit,\
int I,int J){
  if dep==0 then
    search_best(diffCardlimit-\
    limit);
    return ;
  end if
  for (i,j) in ke_available_set
    if !isok_ke(i,j) then
     continue
    end if
    TMP <- TMP+shun(i,j);
    diff <- getdiff();
    search_dfs_ke(dep-1,limit-diff,\
    i,j);
    recover(TMP)
  end for
  search_dfs_shun(dep,limit,0,2);
}
```

4 Optimization

In our ideas, we focus on getting victory in least turns without considering of other players as their states are invisible. That's a way we thought to selectively abandon some optimizations which contribute relatively less to our decisions so that we are capable to allocate more resources on major work. We consider it an aggressive type of decision-making, while the final bot behaves amazingly well.

4.1 Pruning

As one of the commonest optimizations used in searching algorithms, pruning does work efficiently. First, as the dynamic calculation of $LWT(S,T)$, using $LHT(S)$ to prune while searching is available. Then, we limit $LHT(S,T)$ to 6 while searching. This restrict proves to be reasonable. The upper bound of $LHT(S)$ which is actually 7 is easy to know because of the existence of Fan "seven pairs". Using simple statistics, we know that the possibility of a pattern's absence of pairs is less than 1%. Then we can design a DP algorithm to work out the possibility of a pattern whose $LHT(S)$ is more than 6 (equal to 7) as less than 0.5%. For this part of 0.5%, we decide to discard wind tiles or cardinal tiles first for the efficiency of winning in case the program hasn't found a winning hand yet. In the actual running, we find that program can produce all the winning hands in no more than 2 seconds and in less than 1 second if $LHT(S) < 6$, that's what proves that we manage to step to our goal.

4.2 Expanding the Executing Period of Searching

The remaining problem is to resolve the time exceeding when $LHT(S) = 6$. In view of the short distance from our purpose, expectation analysis which may do harm to the accuracy seems unnecessary. So, it occurs to us that not only when needed to response the information we can do some searching but also when there is little need for response (such as when other players implement a claiming operation). In this way we make as full use as possible of all resources.

Here are the concrete measures. We upload the current $LHT(S)$ and $P(S)$ to the platform so that we can download them the next executing time. If we haven't completed the current searching yet when time is exceeding, we will upload the incomplete searching path as well and use the combination of last uploaded and current $LHT(S)$ and $P(S)$ to calculate $E(S)$. Considering that the last and the current results differ slightly, we are confident that this method works.

Another restriction is that the space limit of uploading is 100K, so the saved number of elements in P(S) is considered. We set it as 4,000, which is actually greater than the number of elements in $P(S)$ when $HT(S) \leq 5$, so a quite few of winning plans will be abandoned when $LHT(S) = 6$, which influences quite slightly, and this usually happens only in the beginning 5 turns of the game.

All the same, there is a tiny probability of missing winning plans. We define an easy handler to avoid crashing. We calculate a simple weight of each tile. If the tile exists two times in the pattern, its weight will be added by w_p. And if two tiles form have a chance to form a sequence, their weights will be added by w_s. Initially all tiles' weights are 0. After calculating, we choose the tile of minimum weight to discard, and wind tiles and cardinal tile in priority if we have multiple choices.

4.3 Adjusting Parameters

With the optimizations above, our program manages to achieving our goal. To keep the robustness of the program, we should adjust the related parameters through observing the bot's behavior in the real contests. After some jobs, the final parameters are listed in the following:

Parameters	Values
p_{p2}	$20/255^2$
p_{p3}	$50/255^3$
p_{c2}	$6/255^2$
p_{c3}	$18/255^3$
p_7	$0.7/255$
w_p	3
w_s	1

References

[1] Zhang J , Zhao J , Bai S , et al. Applying speech interface to Mahjong game[C]// Multimedia Modelling Conference, 2004. Proceedings. 10th International. IEEE, 2004.