AI Mahjong

Wan Jing Loh
December 11th 2009
wl246@stanford.edu

1. Introduction

Games like chess and checkers offer a controlled environment with fixed rules and an easy way to compute performance (with their built-in victory conditions), making them ideal test cases for machine learning techniques. As games generally allow for several players, by pitting the computer against human players, one can see how the algorithm measures up against competition. Computers have been acknowledged to be world class players in various games ranging from chess, backgammon and scrabble, while their performance is getting better in more complex games like Go.

In this project, I attempted to develop a viable AI Mahjong player, by building a hand evaluator that is able to accurately calculate how many tiles from victory the player is.

2. The Game of Mahjong & Problems with applying AI techniques to it

Mahjong is a 4–player game which is a common past-time in Asia, especially in China and Japan. Over time, many variant (houserules) have developed, which vary from region to region. In the interests of space, please consult the references for a link to the rules of the game. The rest of the paper will assume basic familiarity with mahjong rules.

Most machine learning techniques focus on two player games where each player takes turns in order. This is true of most of the popular games that have been widely studied, with the possible exception of poker. In mahjong, besides the fact that there are 4 players, it is possible for the player's turn to be 'skipped' due to other players doing melds, which makes it hard to model.

The problem of hand value varying from game to game (and indeed within a game) makes it hard to come up with a general way to evaluate a hand. This is particularly true in the case with a ping-hu (winning hand with all straights), which has many rules in order to be valid. For example, a ping-hu requires the player to be waiting for at least two tiles in order to win off another player's discard. This variability makes it hard to check if a hand is actually a winning hand, and the actual point value of the hand.

Lastly, draws are random, so luck plays a significant role in the game. A large number of games must be played to get an accurate result of an algorithm's strength. For example, whether to go for a easily made hand with low point value, or a higher point hand that may take longer to form. A max payout hand is worth 16 times the payout of a 1 point hand and a lucky draw could easily skew results.

3. Literature Review

Arthur Samuel's checkers playing program paved the way by using reinforcement techniques in game playing. Expanding on his work, the chess playing program Deep Blue defeated the world champion Gary Kasparov in 1996 using a mixture of brute force search and an evaluation function of the board position that has been tuned by analyzing thousands of games.

While brute force search was sufficient for games like chess, it proved intractable in games like backgammon, where the randomness of dice rolls led to a large branching factor. In these games, it is far more important to accurately evaluate the board position than to be able to look many steps into the future. Previously, most evaluation metrics and the features were hand-crafted by experts in the games, with the weights then being fine-tuned by playing many games against itself. G. Tesauro used combined multilayer neural networks along with temporal difference learning to build a backgammon player that learns by playing itself. This program was able to play at world-class level and overturned many of the conventional concepts about how to best play backgammon.

4. Requirements for a good AI Mahjong Player

a) The ability to look at a hand of tiles and judge what kind of hand to form with it

This is the most important requirement. Given tiles, we have to decide what hand to aim for. Mahjong has a large variety of hands and choosing the most likely hand will increase our chances of victory. Likelihood can be determined in many ways, one is the number of extra tiles we need for victory. The second is the number of tiles remaining that will improve our hand. We may only need 1 tile to win, but if all 4 of that particular tile have been discarded, it will be a pointless wait. This is hard to quantify however, as the closer to victory we get, the fewer good tiles will improve our hand. In addition, the value of the hand is also important. We want to maximize the expected value of the hand, not just the probability of success, a 30% chance of a 20 point payoff is likely to be better than a 50% chance of a 10 point payoff. However, we are unlikely to play an infinite number of games, so a 0.01% chance of a million point payoff is something we shouldn't really aim for.

b) The ability to choose when to meld

Draws are random and based on luck, but melding allows us to get specific tiles from the discards of other people. Tile combinations that allow for melding effectively increase the chance of us getting good tiles. However, melding provides information to our opponents and restricts our options, so weighing the trade off is important.

c) The ability to choose the tile to discard

With the ability to evaluate a hand, it might seem simple to choose a tile to discard, we simply choose one of the many tiles that are not useful in forming our hand and discard it. The reality is not so simple, some tiles are better to discard than others. In the early stages of the game, it is often possible to have 2 or more relatively viable hands. It is usually a good idea to discard tiles that keep both hands viable and look at our next few draws before finally deciding on the hand we want to make.

We would also like to discard tiles that do not result in an opponent winning as the person that dealt the winning tile has to play double the points of the hand. This may require us to sacrifice winning and play for a draw in order to not deal into an opponent's hand.

d) The ability to evaluate an opponent's hand

An opponent's discard can tell us much about what hand he is making and how close he is to success. If an opponent has not discarded tiles from the Tong suit since the start of the game, and suddenly discards one now, it is likely he is close to success with a FullColor Tong hand. Discarding Tong tiles now would be very risky and may lead to the loss of lots of points.

My main focus in this project is to get an accurate hand evaluator that looks at a hand and judges how easily winning hands of particular hand types can be put together

5. The Hand Evaluator

The structure of the problem naturally lends itself to a Constraint Satisfaction Problem (CSP). We have several constraints, such as the fact that we must use all 14 tiles in our hand, there must be 4 triplets and a pair, and the fact that the combination of triplets and pairs must satisfy a valid mahjong hand.

In poker literature, the easiest way to check for a poker hand is to convert the hand into a histogram with bins dependent on which hand we are searching for. For example to find a 4 of a kind, the histogram bins are the rank of the cards, and if a bin has 4 cards in it, we have a 4 of a kind. To check for a straight (5 consecutive tiles), we look for 5 consecutive non-empty bins.

Simply applying the poker method to mahjong does not work. Poker hands have relatively simple patterns that are easily detected by the histogram method. Mahjong hands have much more variation in them. In addition melds are fixed, for example, Pungs of 3Suo, 4Suo and 5Suo could not be used as 3 sets of 3,4,5 instead. However, notice that the triplets and pair that make up the hand can be easily detected using the histogram method, if we had a good way of dividing the mahjong hand.

The fact that we have triplets and a pair poses an interesting asymmetric problem in dividing up the hand. Finding all combinations of triplets and pairs was problematic. The problem was solved by introducing an imaginary 15th tile. Now we simply have 5 triplets and can apply the same histogram method to all of them, to see how many tiles satisfy the requirements. The input of the algorithm is the tiles in hand and the melded triplets. The output of the algorithm is a score for each type of hand indicating the number of tiles that are not in conflict with the constraints. A score of 14 indicates a winning hand.

6. Detecting Conflicts – the Histogram Method

We have 3 tiles, one of which may be imaginary, and we wish to figure out how many of those 3 tiles actually aid in forming the hand. We do this by forming a histogram of 56 bins, with each bin representing a single type of tile. Some bins are actually unused and act as dividers. Having these always empty bins also simplifies searching when we have a suit restriction.

We fill the histogram with all three tiles (2 if there is an imaginary tile). If we are looking for a 3 of a kind, we simply find the bin with the largest number of tiles in it and return the number. For consecutive tiles, we look for the largest consecutive number of non-empty non-honor bins. The always empty bins help prevent the search from crossing suit boundaries. I also searched for patterns that consists of filled, empty, filled bins., since that represents a consecutive pattern missing a middle tile.

Suit requirements, such as for a halfcolor hand, are checked by limiting the area of the search.

We set the kind of triplet constraints based on the hand we are currently checking. A halfcolor hand for example, would limit the search to just that particular suit and return the maximum of the results of 3-of-a-kind and consecutive tiles. If there is an imaginary tile, we only accept the result of the 3-of-a-kind search, since we are looking for a pair.

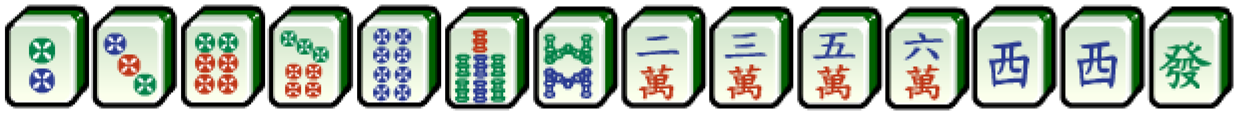
7. Discarding Tiles

Currently, a simplistic model is implemented. We take the most likely hand, and based on the triplets generated, we can tell which tiles are not being used in forming the hand. We then choose one of them to discard.

8. The Simulator

I created a simulator that supports an arbitrary number of human and/or AI players. In 4-players mode, the game behaves just like the real physical game with all the rules faithfully represented. This allows me to compare the evaluation function against my own thoughts on the hand. This portion of the program actually took up a large portion of my time, but will pay dividends in the future when I expand on the code.

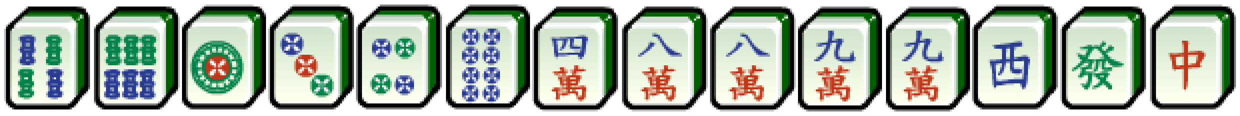
9. Interesting Points

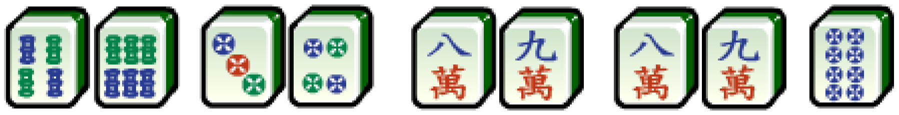


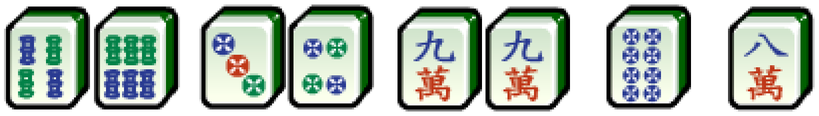
I initially evaluated this hand as having a score of 11 (3 tiles from completion) for a ping-hu, while the hand evaluator evaluated it as only 10 tiles. Clearly the 3 honor tiles are worthless in this case, but where was the last conflicting tile? Each suited tile has at least one neighbor, so what was wrong?

It turns out the structure of the triplets is to blame. While we do have 5 nice sets of at least 2 tiles, none of them can form a pair. As a result, we would eventually have to break one of them up into a pair in order to form winning hand, so we actually need 4 tiles and not 3 to win.

This hand shows the effectiveness of the hand evaluator, as it can spot things that a human might otherwise miss.







I evaluated this hand as having a score of 9 tiles with the relevant 9 tiles shown in the center arrangement. Any non-honor tiles may act as one part of a pair. The hand evaluator scored it with a score of 8, with the relevant tiles being the tiles at the bottom. Due to the evaluator making the 9Wans as a pair, it was unable to make the 8,9 sets, without using at least 2 swaps to have an improvement in score. This example proved that local optima exist. Reshuffling and recalculating fixed this problem.

10. Future Improvements

Currently the hand evaluator can accurately calculate the tiles needed for victory. However, it does not account for the availability of those tiles as well as the value of the hand when making decisions on which hand to play. Incorporating these will make the evaluator better able to distinguish between similar hands and make it able to decide when to meld better. In addition, using temporal difference learning to make the evaluator better able to judge the tradeoffs between hand value and hand speed would be very useful.

The next big step would be to use a Hidden Markov Model to guess at the opponents hand and decide which tile to discard based on the relative value of your hand and the opponents hand.

11. Conclusion

This has been a very fun project and I learnt a lot. It clearly demonstrated to me that properly formulating a problem can reduce the complexity of an algorithm by a lot. However, there is still a lot to be done in order to a an AI that plays mahjong decently.

12. References

1. Samuel, A. Some studies in machine learning using the game of checkers. *IBM J. of Research and Development 3,* (1959), 210-229.
2. Tesauro, G. A parallel network that learns to play backgammon
3. Tesauro, G. Programming Backgammon using self-teaching neural nets*. Artificial Intelligence 134 (2002) 181–199*
4. B. Sheppard . World-championship-caliber Scrabble. *Artificial Intelligence 134 (2002) 241–275*
5. http://en.wikipedia.org/wiki/Mahjong *Rules of the game*

13. Appendix - Code Description of Hand Evaluator

For each type of hand,

1. Melded tiles are checked to make sure they don't conflict with the hand, for example a Pong (3 of a kind) will exclude us from forming a pinghu hand.
2. The sum of the remaining tiles + the imaginary tile will be a multiple (x) of 3
3. These tiles are used to form x Triplets, which are the constraints for the problem
4. The triplets stay fixed, there is one Triplet for the first 3 tiles, 1 for the next 3 and so on.
5. A Triplet's constraints is defined when created, we can enforce that the triplet should be 3 consecutive tiles, 3 of a kind, the suit the tiles must have, etc. This is dependent on the type of hand
6. We can check how many tiles within the triplet satisfy the constraint (maximum of 3, since there are 3 tiles) through the histogram method described in part 6
7. For the Triplet with the imaginary tile, a maximum of 2 tiles can satisfy the constraint (the imaginary tile does not fulfill any requirements and does not appear in the histogram).
8. We then do local search by attempting to swap tiles between Triplets such that the total number of tiles that are not in conflict are increased.
9. We end when no swaps decrease the number of conflicted tiles
10. The method is susceptible to local optima, so we shuffle the tiles randomly and repeat step 7, until the maximum does not change for 5 reshuffles.