Building a Computer Mahjong Player Based on Monte Carlo Simulation and Opponent Models

Naoki Mizukami and Yoshimasa Tsuruoka
Graduate School of Engineering
The University of Tokyo
7-3-1 Hongo, Bunkyo-ku, Tokyo, Japan
Email: mizukami,tsuruoka@logos.t.u-tokyo.ac.jp

***Abstract*—Predicting opponents' moves and hidden states is important in imperfect information games. This paper describes a method for building a Mahjong program that models opponent players and performs Monte Carlo simulation with the models. We decompose an opponent's play into three elements, namely, *waiting*, *winning tiles*, and *winning scores*, and train prediction models for those elements using game records of expert human players. Opponents' moves in the Monte Carlo simulations are determined based on the probability distributions of the opponent models. We have evaluated the playing strength of the resulting program on a popular online Mahjong site "Tenhou". The program has achieved a rating of 1718, which is significantly higher than that of the average human player.**

I. Introduction

In artificial intelligence research, imperfect information games provide challenging and important research questions since such games require one to address common problems in a real world, in which players deal with hidden information and stochasticity. Mahjong is a traditional imperfect information game played in many Asian countries. Among many variants of Mahjong, we focus on Japanese Mahjong, which is one of the most popular table games in Japan. From an AI research point of view, Japanese Mahjong is a challenging game because (a) it is played with more than two players, (b) it is an imperfect information game, and (c) the number of information sets is much bigger than those of popular card games such as poker.

A two-player version of Texas Hold'em, one of the most popular poker variants, has recently been solved by off-line computing of an approximate Nash equilibrium strategy [1]. However, this type of approach is not always feasible since the computational cost for obtaining Nash equilibrium strategies can be prohibitive in certain games. Another approach for building computer players for imperfect information games is using *opponent models*. For opponent modeling, Van der Kleij [2] clusters players based on their playing style using game records. A hand rank distribution is updated in the course of games [3]. In computer Skat, Buro et al. [4] proposed a method that estimates an arbitrary hypothetical world when a player decides moves in the current position using game records. The hypothetical world consists of distributing cards between the unobserved hands of opponents. As the result of training on data from average human players and inference on high-level features of worlds, their Skat-playing program plays at the level of experts human player.

In this work, we build a computer Mahjong player using opponent models for predicting three abstracted elements of an opponent, namely, *waiting*, *winning tiles* and *winning scores*. First, we train the three prediction models using game records of expert human players. We then build a computer Mahjong player that determines its move by using these prediction models and Monte Carlo simulation. We present the performance of our program against human players and a state-of-the-art program.

This paper is organized as follows. Section II describes basic rules and terms of Mahjong. Section III presents what we call one-player Mahjong moves. Section IV, V and VI describe prediction models about waiting, winning tiles and winning scores. Section VII describes how to decide moves using prediction results and Monte Carlo simulation. Section VIII shows some experimental results. Section IX describes related work. Finally, the conclusion is presented in section X.

II. Basic Rules and Terms of Japanese Mahjong

This section describes basic rules and terms of Japanese Mahjong (which we simply call Mahjong in what follows). Mahjong is a table game in which four players start with a score of 25,000 points and compete to achieve the highest score. One game of Mahjong usually consists of four or eight rounds. A player can win a round by completing a winning hand consisting of 14 *tiles*, and get a score according to the hand. At each turn in a round, a player picks up a tile from the *wall* (a set of tiles shared by all players), or picks up a tile discarded by another player, which is called *stealing*. He then discards a tile or declares a win. When a player picks up a winning tile himself, it is called *winning-from-the-wall*, and the other three players share the responsibility of paying out the score. A player can call out *ron* when one of the other players discards a winning tile. It is called *winning-by-a-discard*, and the player who has discarded the tile pays the whole score to the winner.

Mahjong is played using 136 tiles, comprising 34 distinct kinds with four of each kind. One hundred and eight of them are *number* tiles. Each has a number from one to nine and one of three *suits* (*numbers*, *balls*, and *sticks*). The rest are *honor* tiles with no particular ordering. Honor tiles comprise *wind tiles* (east, south, west and north) and *dragon tiles* (white, green and red).

Each player usually tries to make a hand of four *sets* (*melds*) and one pair of tiles. There are two kinds of set, a

978-1-4799-8622-4/15/$31.00 ©2015 IEEE

pung, three identical tiles, and a *chow*, three consecutive tiles of the same suit.

Here, we describe some basic Mahjong terms particularly relevant to this work.

Waiting
: A player is *waiting* if his hand needs only one tile to become complete.

Riichi
: A player can declare *riichi* when he is waiting with no melds made by stealing. Once a player declares riichi, he must discard every tile he picks up except when it is a winning tile.

Folding
: A player *folds* if he gives up to win and only tries to avoid discarding a winning tile for other players. Unlike poker, players do not explicitly declare a fold, and thus folding is not an action but a strategy in Mahjong.

III. One-player Mahjong moves

This section gives a brief explanation of what we call one-player Mahjong moves, which play an important role in our Monte Carlo simulation. First, we define one-player Mahjong as a game of Mahjong in which there is only one player. The goal of the player in a one-player Mahjong game is to complete his hand.

In our previous work [5], we developed a computer program for one-player Mahjong by supervised machine learning. Given a set of 14 tiles, the program can select the move (i.e. tile) that is most likely to lead to quick completion of the hand. Always playing such moves in a standard (four-player) Mahjong game is not a good strategy because you never make a meld by stealing. We therefore extended the program with a machine learning-based component that makes judgments on stealing. In this paper, we call the moves chosen by this extended program *one-player Mahjong moves*.

Playing such one-player Mahjong moves still does not result in strong Mahjong play, because you never fold and often discard a winning tile for other players. In this paper, we address this problem by combining one-player Mahjong moves with moves selected by Monte Carlo simulations, using three prediction models for opponent modeling. The main idea is to achieve a good balance between quick completion of a hand and minimizing the expected loss from discarding a winning tile for other players. In the following sections, we describe our prediction models in detail.

IV. Prediction of waiting

This section describes how to train the model that predicts whether an opponent is waiting or not. In Mahjong, a player poses no immediate risk to other players, if he is not waiting. Accurate prediction of waiting is therefore important in building a strong Mahjong program.

A. Game records for training

We take a supervised machine learning approach for building the prediction model. We use game records in the "Houou table" at the internet Mahjong site called Tenhou[1] as the training data[2]. Only the top 0.1% of the players are allowed to play in the Houou table, so we consider the quality of those game records to be that of expert human players.

For each state in the game records, we generate a binary label indicating whether the opponent's hand is waiting or not, and a feature vector from the other players' points of view. The total number of states is about 1.77×10^7.

TABLE I. Features of waiting prediction

Features	Number of features
Riichi	1
Number of revealed melds and discarded tiles	$5 \times 19 = 95$
Number of revealed melds and turns	$4 \times 19 = 76$
Number of revealed melds and changed tiles[3]	$5 \times 19 = 95$
Number of revealed melds and last discarded tiles	$5 \times 37 = 185$
Kinds of revealed melds and discarded tiles	$136 \times 37 = 5032$
Discard *bonus tiles*[4] and kinds of bonus tiles	34
Discard red bonus tiles[5]	1
Combination of two discarded tiles	$37 \times 37 = 1369$

B. Training the model

We use logistic regression to build a prediction model. The probability that an opponent is waiting in a given state is computed as follows:

$$P(p = waiting) = \frac{1}{1 + \exp\left(-\mathbf{w}^{\mathrm{T}}\mathbf{x_p}\right)}, \quad (1)$$

where $\mathbf{x_p}$ is a feature vector representing the information about an opponent player p and $\mathbf{w}$ is the weight vector for the features. Table I shows the features used in the model. Examples of the features include kinds of tiles an opponent player has discarded and the number of revealed melds and turns. The total number of features is 6,888.

The training of the prediction model is performed by minimizing the following objective function:

$$L(\mathbf{w}) = -\sum_{i=1}^{N}(c_i P(\mathbf{X_i}) + (1-c_i)(1-P(\mathbf{X_i})) + \frac{\lambda|\mathbf{w}|^{\mathbf{2}}}{N}, \quad (2)$$

where N is the number of training examples, $\mathbf{X_i}$ is the i-th training sample, c_i is the binary (1 or 0) label indicating whether the opponent is waiting or not and λ is the regularization term, which is used for alleviating the problem of overfitting to the training data. We set the regularization term λ to 0.01.

We use the FOBOS algorithm [6] to train the weight vector. The weights are updated by using Adagrad [7]. The update equation is as follows:

$$w_{t+1,i} = w_{t,i} - \frac{\eta g_{t,i}}{\sqrt{1 + \sum_{k=1}^{t} g_{k,i}^2}}, \quad (3)$$

where $w_{t,i}$ is the i-th elements of the weight vector, η is the learning rate, t is the number of updates and g is the (stochastic) gradient of the objective function. We set the learning rate η to 0.01.

[1] http://tenhou.net/

[2] The game records are the records of the games played between 20/02/2009 and 31/12/2013

TABLE II. EVALUATION OF WAITING PREDICTION

Player	AUC
Expert player	0.778
Prediction model	0.777
-Discarded tiles	0.772
-Number of revealed melds	0.770

C. Accuracy of prediction

We examine the accuracy of the prediction model described in the previous by using the Area Under the Curve (AUC) as the measure for evaluation. We created the test data by randomly selecting 100 states from the game records (only one state has been sampled from each round). We did not include the states where the target opponent player has already declared riichi since the waiting prediction is straight-forward in such states.

Table II shows the evaluation results. The "Expert player" in the table is an expert human player who is allowed to play in the Houou table. The results show that our prediction model has roughly the same prediction ability as the expert player. We also show the performance achieved by the prediction models that do not use a certain type of features, just below the"Prediction model".

V. PREDICTION OF WINNING TILES

This section describes how to train the model that predicts opponents' winning tiles. In Mahjong, the player who has discarded the opponents' winning tile pays the whole score to the opponent, i.e., it is very damaging to the player. It is therefore important for a program to be able to predict opponents' winning tiles accurately.

A. Game records for training

We use the same game records as the ones described in Section IV. For each kind of tiles, we generate a binary value indicating whether it is a winning tile or not, and a feature vector representing the information available to the other players. The total number of states is about 7.24×10^7.

B. Training for predicting winning tiles

In general, there are one or more winning tiles for an opponent. We therefore address this task as a binary prediction (i.e. a winning tile or not) problem for each kind of the Mahjong tiles, and build 34 prediction models using logistic regression.

Table III shows the features used in the model. Examples of the features include the number of each kind of tiles and the tiles the opponent has already discarded. The total number of features is 31,416.

We train the prediction models in the same fashion as we did for the waiting prediction models described in the previous section. The objective function is

$$L(\mathbf{w}) = -\sum_{i=1}^{N}(c_i P(\mathbf{X_i}) + (1-c_i)(1-P(\mathbf{X_i})) + \frac{\lambda|\mathbf{w}|}{N}, \quad (4)$$

where N is the number of training examples, $\mathbf{X_i}$ is the i-th training sample, c_i is the binary (1 or 0) label indicating whether the tile is a winning tile or not and λ is the regularization term. We use l_1-regularization to achieve sparsity in the weight vector.

We used FOBOS and Adagrad for optimization. We set the learning rate η to 0.01 and the regularization term λ to 0.01.

TABLE III. FEATURES OF WINNING TILE PREDICTION

Features	Number of features
Number of tiles a player can count	$5 \times 34 = 170$
Safety tiles[6]	34
Changed tiles n times	$3 \times 34 = 102$
Bonus tiles	34
The number of turns and tiles	$18 \times 37 = 666$
Tiles a player discard when a player declares riichi	37
Kinds of revealed melds and discarded tiles	$136 * 37 = 5032$
Combination of kinds of revealed melds	$136 * 136 = 18496$
Combination of discarded tiles and whether changed tiles or not	$37 \times 37 \times 2 \times 2 = 5476$
Discarded tiles and changed tiles	$37 \times 37 = 1369$

TABLE IV. EVALUATION OF WINNING TILE PREDICTION

Player	Evaluation value
Expert player	0.744
Prediction model	0.676
-Revealed melds	0.675
-Discarded tiles	0.673
Random	0.502

C. Accuracy of prediction

Evaluation of the model for predicting winning tiles should reflect the rules of Mahjong. We need to predict all winning tiles for opponents accurately since discarding any of them will lead to the losing of the round. We therefore perform the evaluation of the models as follows. The tiles that a player has are arranged in ascending order of probability of being a winning tile. The evaluation value is computed as follows:

$$Evaluation\,value = \sum_{i=1}^{N} \frac{Clear_i}{AT_i - WT_i}, \quad (5)$$

where N is the number of test samples, $Clear_i$ is the smallest rank among the winning tiles, $AT_i\,(All\,Tiles)$ is the number of the kinds of the tiles the player has and $WT_i\,(Winning\,Tiles)$ is the number of the kinds of the winning tiles.

We created the test data by randomly selecting 100 states from the game records (only one state has been sampled from each round). Table IV shows the evaluation results. *Random* is a player that discards tiles randomly. The results show that the performance of the prediction model is considerably higher than that of Random but is lower than that of an expert player.

VI. PREDICTION OF SCORES

This section describes how to train a model that predicts the score that the player has to pay if the tile he discards is a winning tile for an opponent. The model for the case of winning-from-the-wall is trained analogously.

[3]A player discards the tile that has existed in his hand

[4]If a player wins with *bonus tiles*, the score increases according to the number of those.

[5]We use marked red number 5 tiles that also count as bonus tiles.

[6]The tiles a player has already discarded.

TABLE VI. EVALUATION OF SCORE PREDICTION

Player	Mean square error
Prediction model	0.37
-Revealed Melds	0.38
-Revealed fan value	0.38
Expert player	0.40

A. Game records for training

For each state where a player has won in the game records, we generate a training sample by using the score and creating a feature vector representing the information available to the other players. The total number of states is about 5.92×10^7.

B. Training for predicting scores

In Mahjong, the score of a hand basically increases exponentially according to the *fan*[7] value of the hand. We therefore build a linear regression model that predicts the natural logarithm of an actual score in the training data. Table V shows the features used in the model. Examples of the features include the fan value of the revealed melds and the number of revealed bonus tiles. The total number of features is 26,889.

The model predicts $HS\,(Hand\,Score)$ in a given state as follows:

$$HS = \mathbf{w}^{\mathrm{T}}\mathbf{x}, \tag{6}$$

where $\mathbf{x}$ is a feature vector and $\mathbf{w}$ is the weight vector for the features.

The training of the prediction model is performed by minimizing the following objective function:

$$L(\mathbf{w}) = \sum_{i=1}^{N}(HS_i - c_i)^2 + \frac{\lambda|\mathbf{w}|^{\mathbf{2}}}{N}, \tag{7}$$

where N is the number of training examples, HS_i is the predicted value, c_i is the natural logarithm of the actual game score and λ is the regularization term. When using this model, the output of the model is exponentiated to obtain the actual score in the original scale. We use FOBOS and Adagrad to train the weight vector. We set the learning rate η to 0.01 and the regularization term λ to 0.01.

C. Accuracy of prediction

We evaluated the model using mean square error between the predicted value and the actual score. We created the test data by randomly selecting 100 states from the game records (only one state has been sampled from each round). We give human subjects information about the tiles the players have discarded and who wins. The evaluation results in Table VI show that the performance of the prediction model is even higher than that of an expert player.

[7]*Fan* is the sum of the values of winning hands.
[8]Tiles are three adjacent numerical to safety tiles.
[9]A hand consisting of only 2 through 8 number tiles.
[10]Dragons tiles, a player's own wind tiles and wind of the round tiles.
[11]A hand consisting of only one suit and honor tiles
[12]A hand consisting of only pungs and a pair.

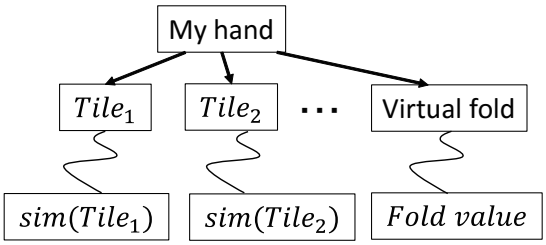

Fig. 1. Overview of Monte Carlo moves

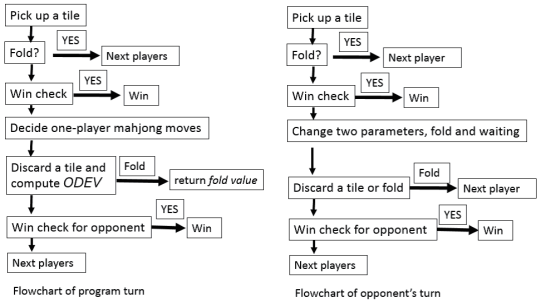

Fig. 2. Flowchart of each player

VII. MONTE CARLO MOVES USING OPPONENT MODELING

This section describes how our program determines its move (action) by using the prediction models and performing Monte Carlo simulations. The evaluation results given in the previous sections demonstrate that the performance of the prediction models for the three kinds of states of opponents' hands is comparable to that of expert players. We use the models to estimate an expected value that our program pays.

We define $LP\,(Losing\,Probability)$ as the probability that an opponent p is waiting and $Tile$ is the winning tile for him and it is computed as follows:

$$LP(p, Tile) = P(p = waiting) \times P(Tile = winning), \tag{8}$$

where p is an opponent player and $Tile$ is the tile the program discards. $EL\,(Expected\,Loss)$ is LP multiplied by HS and computed as follows:

$$EL(p, Tile) = LP(p, Tile) \times HS(p, Tile). \tag{9}$$

The score that the program is expected to obtain by discarding a tile is computed as

$$\begin{aligned} Score(Tile) = Sim(Tile)\times & \\ \prod_{p\in opponents}(1 - LP(p, Tile))- & \\ \sum_{p\in opponents} EL(p, Tile), & \end{aligned} \tag{10}$$

where $Sim(Tile)$ is a game score computed using Monte Calro simulation and the second term is the expected loss the program has to pay when discarding $Tile$. The program calculates $Score(Tile)$ for each tile in its hand and selects the tile that has the highest score. We call the moves that are chosen in this way *Monte Carlo moves*.

TABLE V. FEATURES OF SCORE PREDICTION

Features	Number of features
Riichi and dealer	$2 \times 2 = 4$
Revealed fan value and countable bonus tiles	$7 \times 8 = 56$
Riichi or the number of revealed melds and revealed a fan value and countable bonus tiles	$3 \times 7 \times 8 = 168$
Kinds of revealed melds and revealed a fan value and countable bonus tiles	$136 \times 7 \times 8 = 7616$
Combination of two kinds of revealed melds	$136 \times 136 = 18496$
The number of revealed melds and revealed a fan value and countable bonus tiles	$5 \times 7 \times 8 = 280$
Riichi and discarded tiles is suji[8]or all simples[9]	$3 \times 2 \times 2 = 12$
Steal non value honor[10]tiles and steal value honor tiles, the number of revealed melds is zero	3
Discarded tiles is bonus tiles, one or two adjacent numerical, same suit, no relation	5
A hand can create all simples and bonus tiles and countable bonus tiles	$2 \times 2 \times 8 = 32$
A hand can create flush[11](honitsu and tinitsu) revealed a fan value and bonus tiles is same suit	$5 \times 7 \times 2 \times 2 = 140$
A hand can create all pungs[12]and revealed a fan value and bonus tiles is same suit	$5 \times 7 \times 2 = 70$
Number of revealed melds of dragon tiles	3
Number of revealed melds of wind tiles	4

Now we describe how we actually compute the values of $Sim(Tile)$ by Monte Carlo simulation. Figure 2 shows two flowcharts representing what are done in the program's and the opponents' turns. In the program's turns, the program plays a one-player Mahjong move. In opponents' turns, they win or declare riichi based on some probability distributions. The discarded tiles are computed by assuming that the opponent players pick up a tile from the wall and simply discard it right away. We carry out the same number of simulations for each tile. The same random numbers are used for controlling opponents' moves and for generating the wall to minimize the effect of randomness in simulation.

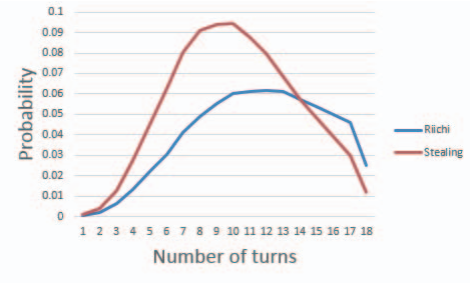

Fig. 3. Probability of waiting player of each turn

A. Opponents' turn

In opponents' turns, they decide their moves based on a probability distribution without considering their hands. Instead, each of them has two binary parameters indicating whether he is waiting or folding.

First, we describe the waiting parameter. An opponent can win only when he has been waiting and picked up a winning tile from the wall or someone has discarded a winning tile for him. The initial value of the waiting parameter is a set to the waiting probability predicted by the model given in section IV in current states. The waiting parameter during the simulation is determined when an opponent player picks up a tile based on another probability distribution. We call the probability $WPET\,(Waiting\,Probability\,at\,Each\,Turn)$ and calculate it using one-player Mahjong moves in advance.

We describe how to calculate $WPET$. First, we generate a wall. For each turn, a player discards a tile based on one-player Mahjong moves. The opponent players pick up a tile from the wall and discard it. If a player becomes waiting, the game is over and the turn is recorded. We calculate $WPET$ as follows:

$$WPET_i = \frac{BW_i}{R_i}, \tag{11}$$

where i is the turn number, $BW_i\,(Becomes\,Waiting)$ is the number of times the player becomes waiting in the turn and $R_i\,(Reach)$ is the number of times the player reaches the turn.

There are two kinds of waiting, riichi and stealing. In the case of riichi, a player cannot steal a tile. We calculate $WPET$ by playing 10^6 games. Figure 3 shows the result. The kind of waiting is decided based on the percentage of stealing before the simulation is carried out and is not changed during the simulation. The percentage of stealing is 35%, which is the average of expert players.

Here we describe the folding parameter. If players do not fold, players who declare riichi win easily. This is not a realistic simulation. To solve this problem, we make opponent players decide to fold.

First, we describe how to decide whether to fold or not using a probability. The biggest difference between one-player Mahjong moves and those in game records is folding. The $FP\,(Folding\,Probability)$ is calculated using game records and one-player Mahjong moves as follows:

$$FP(situation) = \frac{different_{situation}}{count_{situation}}, \tag{12}$$

where $situation$ is a combination of whether the program is waiting or not and the number of opponent players who have stolen tiles or declared riichi, $different_{situation}$ is the percentage that top three one-player Mahjong moves are different from those under $situation$, and $count_{situation}$ is the number of $situation$. The total number of situations is $5,739$. The total number of states is about 1.28×10^8.

We describe how to move for folding. Folding in Mahjong differs from that of poker. A player does not need to declare a fold, but he discards tiles. In real games, if a player decides to fold, he can usually avoid discarding winning tiles for the opponents. To emulate the folding situation, a player picks up a tile and discards no tiles as fold in the simulation. We call these moves *virtual folding*. The other players cannot call out ron.

The initial value of the folding parameter is set to *NOT FOLD*. The folding parameter is determined when an opponent player picks up a tile.

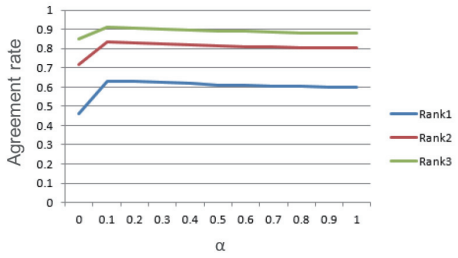

Fig. 4. Agreement rate of each threshold

B. Program's turn

Our program decides moves based on one-player Mahjong moves during the simulation. However, one-player Mahjong moves always aim to win without folding, which results in an unrealistic simulation. To solve this problem, our program decides whether to do a virtual fold or not based on a $ODEV$ ($One\text{-}Depth\,Expected\,Value$). $ODEV$ is an expected value that is calculated by searching game trees without Monte Carlo simulation until the program's next turn. An expected value ($fold\,value$) and the probability of the exhaustive draw[13] (PED) can be calculated using virtual folds. The probability distribution of picking up tiles and opponents' two parameters are fixed while calculating this value. Algorithm 1 describes how to compute $ODEV$. The parameters of this algorithm are the same as those calculated at the root node.

C. Swiching from one-player Mahjong moves to Monte Carlo moves

In opening states, Monte Carlo moves tend to play bad moves, because the time that can be used for simulation is short and every player needs many moves to win. One-player Mahjong moves are more suitable for opening states. We switch from one-player Mahjong moves to Monte Calro moves as follows:

$$\begin{cases} One\text{-}palyer\,Mahjong\,moves & if(\frac{\sum_{p\in opponents} EL(p,Tile)}{fold\,value} \leqq \alpha) \\ Monte\,Carlo\,moves & otherwise \end{cases}, \tag{13}$$

where α is the threshold. The value of α was set so that it maximizes the agreement rate between the moves selected by the program and the ones in the game records.

We used 10,000 states as the evaluation data. Monte Carlo moves are always computed in a second. Figure 4 shows the result. Rank n is the percentage that the top n candidate moves match those in game records. We set α to 0.2 based on the result.

VIII. Experiments

In this section, we present evaluation results on the performance of our program.

[13]No player wins after taking all tiles from the wall.

Algorithm 1 One-Depth Expected Value

function ODEV
  $p = 1$ ▷ Probability of reaching current states
  $value = 0$
  $PWFW = 0$ ▷ Probability of winning-from-the-wall
  $SWFW = 0$ ▷ Score of winning-from-the-wall
  $SWD = 0$ ▷ Score of winning-by-a-discard about the program
  $P(get\,tiles)$ ▷ Probability of getting the tiles
  foreach $i \in all\,players$ **do** ▷ Opponent players and program
    foreach $j \in all\,kinds\,of\,tiles$ **do**
      $PWFW_i+ = P(j = winning) \times P(get\,tiles = j)$
      $SWFW_i+ = HS_{WFW}(i,j) \times P(j = winning) \times P(get\,tiles = j)$
    end for
  end for
  foreach $j \in all\,kinds\,of\,tiles$ **do**
    $SWD+ = HS(program,j) \times P(j = winning) \times P(get\,tiles = j)$
  end for
  foreach $i \in opponents$ **do**
    $value- = EL(i,Tile)$
  end for
  foreach $i \in opponents$ **do**
    $value- = p \times WAITING(Player_i) \times PWFW_i \times SWFW_i$
    $p\times = (1 - WAITING(Player_i) \times PWFW_i)$
    $value+ = p \times (1 - FOLD(Player_i)) \times PWFW_{program} \times SWD$
    foreach $k \in all\,players$ **do**
      if $k \neq i$ **then**
        $p\times = (1 - PWFW_k \times (1 - Fold(Player_i)))$
      end if
    end for
  end for
  $value+ = p \times SWFW_{program}$
  $value- = p \times PED \times fold\,value$
  if $value \leqq 0$ **then**
    return **FOLD**
  else
    return **NOT FOLD**
  end if
end function

function WAITING(Player)
  if $Player.waiting = YES$ **then**
    return **1**
  else
    return **0**
  end if
end function

function FOLD(Player)
  if $Player.Fold = YES$ **then**
    return **1**
  else
    return **0**
  end if
end function

TABLE VII. RANK DISTRIBUTION

	1st	2nd	3rd	4th	Average rank
Our program	0.252	0.256	0.247	0.245	2.48 ±0.07
Mattari Mahjong	0.248	0.247	0.250	0.255	2.51 ±0.07
Mizukami et al. [5]	0.243	0.226	0.222	0.309	2.59 ±0.07

TABLE VIII. RATE OF WINNING AND DISCARDING A WINNING TILE FOR OPPONENTS

	Winning rate	Rate of discarding a winning tile for opponents
Our program	0.222	0.125
Mattari Mahjong	0.200	0.122
Mizukami et al. [5]	0.228	0.178

A. Evaluation with Mattari Mahjong

We compared our program against Mattari Mahjong[14]. To the best of our knowledge, Mattari Mahjong is the strongest among the publicly available Mahjong programs. The evaluation function of Mattari Mahjong is created by the heuristic combination of statics.

The length of a game is four rounds. The moves are computed in a second. Our program plays with three Mattari Majhong opponents. We used a *match game type*, which is a duplicate mode that generates the same walls and hands using the random numbers, and allows us to compare the results of Mattari Mahjong plays and that of our program. Our program plays 1000 games. We use the average rank for evaluation.

Table VII shows the results. We calculated the confidence interval of the average rank (p-value ≤ 0.05) using bootstrap resampling [8]. The difference between the average rank of previous work [5] and that of our program is statistically significant by welch t-test (p-value $= 0.01$). In the case of Mattari Mahjong, the result of our program is better than that of Mattari Mahjong, but the difference is not statically significant (p-value $= 0.29$).

Table VIII shows the percentage of winning and the percentage of discarding a winning tile for opponents. The results suggest the playing strength of our program is higher than that of Mattari Mahjong.

B. Evaluation on Tenhou

To examine the playing strength of our program compared to that of human players, we had our program play on the internet Mahjong site called Tenhou. The rules are the same as the case of Mattari Mahjong. Players can play games at various tables depending on their skill level. There are four types of tables, “Houou”, “Tokujou”, “Joukyuu” and “Ippan” tables in descending order of playing strength. We use the average rank (rating) for evaluation. The rating (R) has a negative correlation to the average rank. The rating is computed as follows:

$$R' = R + (50 - Rank \times 20 + \frac{AveR - R}{40}) \times 0.2, \quad (14)$$

where $Rank$ is the rank of previous games. $AveR$ is the average R of players in the table. The inital R is 1500. An improvement in average rank by 0.1 roughly corresponds to 100 rating points

[14]http://homepage2.nifty.com/kmo2/

TABLE X. RATE OF WINNING AND DISCARDING A WINNING TILE FOR OPPONENTS

	Winning rate	Rate of discarding a winning tile for opponents
Our program	0.245	0.131
Mizukami et al. [5]	0.256	0.148

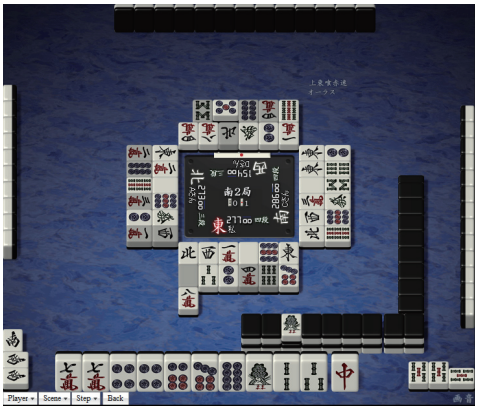

Fig. 5. Fold state

There are two evaluation measures using the rating, the *stable* and *guaranteed* rating[15]. The stable rating is the rating computed by assuming that a player gets current results forever. The guaranteed rating is the lower one-sided bounds of the stable rating when the continuing results of games are arbitrarily selected. This evaluation is used for comparing the results of human players.

C. Results

Table IX shows the result. The difference between the average rank of previous work [5] and that of our program is not statistically significant by welch t-test (p-value $= 0.22$). Our program, however, improves the guaranteed rating. Table X shows the percentage of the winning and rate of discarding a winning tile for opponents.

D. Discussion

Our program has a tendency of folding when its hand is weak and opponent players declare riichi. Figure 5 shows a state in which our program’s decision is to fold. Our program is waiting, but is hard to win. The probability that the red tile is a wining tile for opponents is high. Our program folded and discarded the eight balls.

Our program can play moves that are close to those of human players. Figure 6 shows a state where our program is sitting on the fence. The best move for completing a hand would be 3 or 5 balls in this state. However, they are likely to be winning tiles for opponents. Our program discarded the seven balls. At the end of the round, our program was waiting.

Compared to our previous work, our program improves the result against Mattari Mahjong. On the other hand, it does not improve the result against human players. A possible reason is that the winning ability of Mattai Mahjong is higher than that of human players, and Monte Carlo moves are used more often in the actual games. Folding is better played than before.

[15]http://totutohoku.b23.coreserver.jp/hp/SLtotu14.htm

TABLE IX. RANK DISTRIBUTION

	1st	2nd	3rd	4th	Average rank	Number of games	Stable rating	Guaranteed rating
Our program	0.241	0.281	0.248	0.23	2.46 ±0.04	2634	1718	1690
Mizukami et al. [5]	0.253	0.248	0.251	0.248	2.49 ±0.06	1441	1689	1610

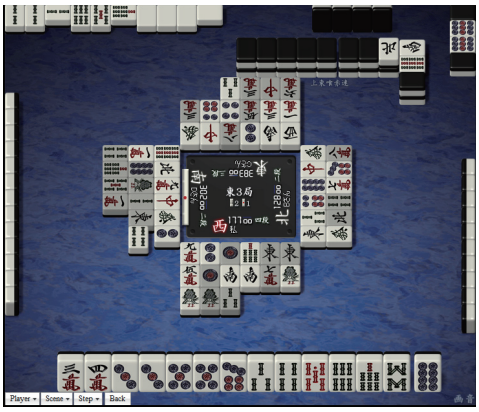

Fig. 6. Fence-sitting state

There are bad moves in one-player Mahjong moves when our program's hand is hard to win. One-player Mahjong moves are used to calculate the percentage of opponents who are waiting or folding, so the improvement of those would make the whole program better. Our program uses some heuristic rules such as the one that if a player is waiting, he must declare riichi. To decide good moves, we need to improve the model for predicting whether a player declares riichi or not.

IX. RELATED WORK

Previously, we defined one-player Mahjong as a Mahjong game where the number of players is one and players cannot steal a tile [5]. We developed a one-player Mahjong player using game records of human players, analyzed the difference between four and one-player Mahjong and extended the one-player Mahjong by filling the difference. The playing strength of our program was higher than that of the average human player. These results suggest that it is important for a program to be able to fold and steal tiles in Mahjong. One of the problems with this approach is that we needed to manually tag data containing state where players fold. A large amount of combination data of the player's hands and situations of a game is required. Another problem is that it is difficult to collect required data for training. Yet another problem is that the program cannot choose *fence-sitting*[16] moves.

Wagatsuma et al. [9] proposed a Support Vector Regression method for predicting the score of an opponent's hand. It extracts feature vectors from game records and predicts values when players discard tiles. One problem with their method is that it is not effectively trained due to some states where the players are waiting in the training data set. Another problem is that their estimation accuracy is only evaluated based on the agreement rate between humans and the program.

Miki and Chikayama [10] proposed Mahjong players using Monte Carlo tree search. Due to the enormous number of leaf nodes for searching game trees of Mahjong, they use Monte Carlo tree search that performs simulations. Opponents' hands and moves are randomly chosen in the simulation. Even though the program does not use knowledge of Mahjong, its playing strength is better than a naive program that only attempts to reduce the number of tiles to win. The playing style of the program differs from that of human players. The program often steals a tile and tries to become waiting, since opponent players do not win in the simulation.

There is a large body of previous work on computer poker. The CFR+ method computes an approximate Nash equilibrium strategy using the results of self-play and has recently solved Heads-up limit hold' em poker [1]. A Nash equilibrium strategy will never lose in the long run. However, it cannot identify and exploit weaknesses in its opponents' play. Van der Kleij [2] used opponent modeling and Monte Carlo tree search for exploitation. Opponent modeling focuses on predicting moves and hands. He proposed a clustering algorithm that clusters players based on their playing style using game records. Aaron [3] proposed a method that updates opponent models in the course of games for exploitation. The program updates a hand rank distribution in the current game state when the showdown occurs.

What makes opponent modeling difficult in Mahjong is that a program usually plays a small number of games with unspecified players. Naive opponent modeling that attempts to predict specific opponents' moves and hands is not suitable for Mahjong. We therefore take a different approach for opponent modeling that treats general players with abstracted hands.

X. CONCLUSION

In this paper, we have proposed a method that considers opponent models trained with game records and decides moves using prediction results and Monte Calro simulation. Experimental results show that the performance of the three prediction models is higher than that of an expert player. Our program achieves a good balance between folding and aiming for a win like human players. On the large online Mahjong site, Tenhou, our program has achieved a rating of 1718 in 2643 games, which is roughly the same as that of the average players in the Joukyuu table.

In future work, we plan to take the scores of the players into account. Our program chooses the same moves regardless of the players' scores. Expert players consider scores when they make a hand and judge whether to fold or not against riichi. These decisions have a great influence on the final rank and are essential in improving playing strength. The value calculated in a simulation needs to be converted into a different value that reflects the result of the game. If the model predicts a rank distribution using current scores, an expected rank can be used as a reward of simulation.

[16] *Fence-sitting* is a strategy in which a player tries to win or make a waiting hand and, at the same time, avoid discarding a winning tile for opponents.

REFERENCES

[1] M. Bowling, N. Burch, M. Johanson, and O. Tammelin, "Heads-up limit hold'em poker is solved," *Science*, vol. 347, no. 6218, pp. 145–149, 2015.

[2] A. Van Der Kleij, "Monte Carlo tree search and opponent modeling through player clustering in no-limit Texas hold'em poker," Master's thesis, University of Groningen, 2010.

[3] A. Davidson, "Opponent modeling and search in poker: Learning and acting in a hostile and un-certain environment," Master's thesis, University of Alberta, 2002.

[4] M. Buro, J. R. Long, T. Furtak, and N. R. Sturtevant, "Improving state evaluation, inference, and search in trick-based card games." in *Proceedings of 21th International Joint Conference on Artificial Intelligence*, 2009, pp. 1407–1413.

[5] N. Mizukami, R. Nakahari, A. Ura, M. Miwa, Y. Tsuruoka, and T. Chikayama, "Realizing a four-player computer mahjong program by supervised learning with isolated multi-player aspects," *Transactions of Information Processing Society of Japan*, vol. 55, no. 11, pp. 1–11, 2014, (in Japanese).

[6] J. Duchi and Y. Singer, "Efficient online and batch learning using forward backward splitting," *The Journal of Machine Learning Research*, vol. 10, pp. 2899–2934, 2009.

[7] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *The Journal of Machine Learning Research*, vol. 12, pp. 2121–2159, 2011.

[8] E. W. Noreen, "Computer-intensive methods for testing hypotheses: an introduction," *Computer*, 1989.

[9] A. Wagatsuma, M. Harada, H. Morita, K. Komiya, and Y. Kotani, "Estimating risk of discarded tiles in Mahjong using SVR," *The Special Interest Group Technical Reports of IPSJ. GI,[Game Informatics]*, vol. 2014, no. 12, pp. 1–3, 2014, (in Japanese).

[10] A. Miki and T. Chikayama, "Research on decision making in multi-player games with imperfect information," Master's thesis, University of Tokyo, 2010, (in Japanese).