Estimating the Probability of Winning for Texas Hold'em Poker Agents

Luís Filipe Teófilo
Departamento de Engenharia Informática, Faculdade de Engenharia da Universidade do Porto, Portugal
Laboratório de Inteligência Artificial e de Ciência de Computadores, Universidade do Porto, Portugal
luis.teofilo@fe.up.pt

Abstract. The development of an autonomous agent that plays Poker at human level is a very difficult task since the agent has to deal with problems like the existence of hidden information, deception and risk management. To solve these problems, Poker agents use opponent modeling to predict the opponents next move and thereby determine its next action. In this paper are described several methods to measure the risk of playing a certain hand in a given round of the game. First, we discuss the game of poker and the expectation in its events. Next, several hand evaluation and classification techniques are described and compared in order to determine the advantages of each one. The current methods to deal with risk management can help the agent's decision. However, in future work, the integration of these methods with opponent modeling techniques should be improved.

Keywords: Poker, Opponent Modeling, Probabilities, Autonomous Agents, Strategy Games, Artificial Intelligence

1 Introduction

Poker is a game that is increasingly becoming a field of interest for the AI research community on the last decade. This game presents a different challenge when compared to other strategy games like chess or checkers. In these games, the two players are always aware of the full state of the game. Unlike that, the Poker's game state is hidden because each player can only see its cards or the community cards and, therefore, it's much more difficult to analyze. Poker is also stochastic game i.e. it admits the element of chance.

The combination of chance with incomplete information makes it essential to model opponents in order to create a competitive player. When the agents identifies the opponents' playing style it can adapt its strategy in order to beat them, by making decisions that have better probability of success [1, 2].

The goal of this work is to determine how an agent can measure the risk of its actions. This article particularly focuses on the game information that is visible to the player: its cards and the community cards. An agent, by knowing how much strong it is, can make better decisions, by managing the risk of betting.

The rest of the paper is organized as follows. Section 2 describes the game of Poker and more particularly the variant that was studied – Texas Hold'em.

Section 3 describes hand odds evaluators, that is, algorithms that are capable of estimating the probability of the player's hand being victorious. Section 4 describes hand rank evaluators which are functions that score a Poker hand which are essential to hand odds evaluators. Finally in section 5 presents the paper main conclusions and some pointers for future work.

2 Texas Hold'em Poker

Poker is a generic name for literally hundreds of games, but they all fall within a few interrelated types [3]. It is a card game in which players bet that their hand is stronger than the hands of their opponents. All bets go into the pot and at the end of the game, the player with the best hand wins. There is another way of winning the pot that is making other players forfeit and therefore being the last standing player.

2.1 Hand Ranking

A poker hand is a set of five cards that identifies the strength of a player in a game of poker. It is not necessary to have all the cards belonging to one player, because in poker there is the definition of community cards – cards that belongs to all players. In poker variations that use community cards, the player hand is the best possible hand combining his cards with community cards.

The possible hand ranks are (stronger ranks first): Royal Flush, Straight Flush, Four of a Kind, Full House, Straight, Three of a Kind, Two Pair, One Pair and Highest Card.

2.2 No Limit Texas Hold'em

No Limit Texas Hold'em is a Poker variation that uses community cards. At the beginning of every game, two cards are dealt for each player. A dealer player is assigned and marked with a dealer button. The dealer position rotates clockwise from game to game. After that, the two players to the left of dealer post the blind bets. The first player is called small blind, and the next one is called big blind. They respectively post half of minimum and the minimum bets. The player that starts the game is the one on the left of the big blind.

After configuring the table, the game begins. The game is composed by four rounds (PreFlop, Flop, Turn, River) of betting. In each round the player can execute one of the following actions: Bet, Call, Raise, Check, Fold or All-In.

In any game round, the last player standing wins the game and therefore the pot. If the River round finishes, the player that wins is the one with the highest ranked hand.

2.3 Opponent Modeling

Since Texas Hold'em is a game of incomplete information, it is essential to model the opponents in order to predict their actions. By predicting the opponents' actions, the player is able to optimize his profit.

One good example of opponent modeling is the Sklansky groups [3] which define types of players and common actions that they take for each group of cards.

3 Hand Odds Evaluation

Evaluating the odds of a hand consists in measuring the quality of the hand in a given round of the game. By evaluating the hand it is possible to determine the probability of winning or losing the current game. By using this knowledge, the agent can decide either to fold the hand or play it, based on the probability of success and the risk that the agent is willing to take.

The hand evaluation function may consider the following variables:

- Player cards;
- Number of opponents;
- Community cards;
- Possible community cards to come;
- Possible opponents cards;

The hand evaluation function returns a number in an interval, typically a real number between 0.0 and 1.0. If it returns the lower limit, this usually means that the hand will lose no matter what happens. Conversely, if it returns the upper limit, this means that victory is mathematically assured – the only way of losing is to unwisely fold the hand.

3.1 Current Hand Odds

The current hand odds algorithm calculates the probability of a player hand being better than the opponents' hands, considering that the opponents can have any remaining non visible cards and that they won't fold until the showdown.

For instance, consider the following game state:

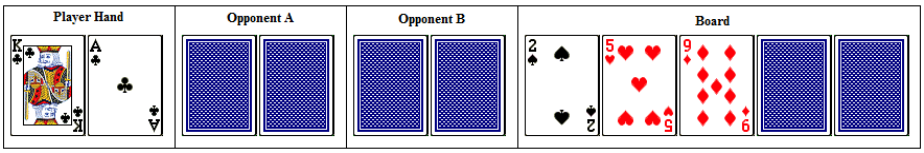

Fig. 1. Game state example with three players in Flop round

To calculate the player's current hand odds for this game state, it must be considered that "Opponent A" cards, "Opponent B" cards and the remaining invisible

board cards can be any cards other than **K♣**, **A♣**, **2♠**, **5♥** or **9♦**. For example, a valid set could be:

- Opponent A: 3♦, 3♥;
- Opponent B: 4♣, 5♣;
- Board: 2♠, 5♥, 9♦, Q♠, Q♣.

The algorithm can be represented by the following code:

```
function CurrentHandOdds (hand, board, numOpponents){
  const fullDeck = [Ac, Ad, As, Ah, Kc, Kd …]
  remainCards = fullDeck – hand – board//set operations
  ahead = behind = 0
  /* for each possible boards */
  for each case(completeBoard =
PossibleBoards(remainCards, board)) {
    handRank = Rank(hand, completeBoard)
    remainOppCards = fullDeck - completeBoard
    /* for each combination of possible cards
    for each case(opponents = Opponents(numOpponents,
remainOppCards)) {
      if (any rank(opponent in opponents,
completeBoard) > handRank) {
         behind++
      } else { ahead++ }
    }
  }
```

As it can be seen in the above code, all possible remaining card sequences are considered. For this reason, this process can be time consuming, given the large number of possible sequences of cards. For the given example (Fig. 1), the number of remaining cards is 47 (52-3-2) and the sequence size is 6. This means that the total number of card sequences is $47\underline{6} = \frac{47!}{(47-6)!} \approx 7.73 \times 10^9$. This would take a lot of time to calculate.

The solution for this problem is to use Monte Carlo Method [4]. The Monte Carlo Method considers a small random subset of card sequences. This means, that the calculation is much faster. The only problem with this technique is that the obtained result is an approximation. However, this experience [5] showed that the error is very small, making this a very reliable solution.

3.2 Hand Strength

Hand strength[6, 7] is the probability that a given hand is better than any other possible hand. It consists in enumerating all possible hands that an opponent can have a checking if hand is better than the enumerated hand. By counting the number of times the player's hand is ahead, it is possible to measure the quality of the hand. It is possible to calculate the hand strength as:

```
function HandStrength(ourcards, boardcards){
  ahead = tied = behind = 0
  ourrank = Rank(ourcards, boardcards)
  /*Consider all two-card combinations of remaining
cards*/
  for each case(oppcards) {
    opprank = Rank(oppcards, boardcards)
    if(ourrank > opprank) ahead++
      else if (ourrank == opprank) tied++
        else behind++
  }
  return (ahead + tied / 2) / (ahead + tied + behind)
}
```

The main advantage of this algorithm is that it works in any round of Texas Hold'em. Moreover, it is capable of giving a quick estimate of the probability of success since the number of cycles in this algorithm is small (1225 at most). It is also possible to calculate the hand strength against a varying number of opponents by raising the found probability to that number (equation 1).

$$\boldsymbol{HS_n = (HS_1)^n} \quad \textbf{(1)}$$

3.2.1 Combining hand strength with opponent modeling

In this article [8], the author suggests that it is possible to combine the hand strength algorithm with opponent modeling, in order to calculate the hand strength taking into account the opponents. To do this, the new algorithm instead of iterating over every possible hand, it only iterates over the cards that the opponent probably has, using Sklansky Groups [3]. This approach was only tested in heads up games.

3.3 Hand Potential

Hand potential[6, 7] is an algorithm that calculates the possible evolution of the hand quality throughout the game. When the game reaches the Flop round, there are still two more board cards to be revealed. This means that the current hand rank might improve; because the hand is composed of a set of five available cards (pocket or community cards) that has the highest rank among all available cards.

This algorithm is very similar to hand strength, but instead of only considering the current available cards, it considers the possible community cards that have not been revealed yet. This algorithm also considers that the opponents hand might improve as well.

Hand potential has two components:

- Positive potential: of all possible games with the current hand, all scenarios where the agent is behind but ends up winning are calculated.

- Negative potential: of all possible games with the current hand, all the scenarios where the agent is ahead but ends up losing are calculated.

The components of hand potential can be calculated as follows:

```
function HandPotential(ourcards, boardcards){
  int array HP[3][3], HPTotal[3] /* Init to 0 */
  ourrank = Rank(ourcards, boardcards)
  /* Consider all two-combinations of remaining cards*/
  for each case(oppcards) {
    opprank = Rank(oppcards, boardcards)
    if(ourrank>opprank) index = ahead
      else if(ourrank=opprank) index = tied
        else index = behind
    HPTotal[index]++
    /* All possible boards to come. */
    for each case(board) {
      ourbest = Rank(ourcards, board)
      oppbest = Rank(oppcards, board)
      if(ourbest>oppbest) HP[index][ahead]++
        else if(ourbest==oppbest) HP[index][tied]++
          else HP[index][behind]++
    }
  }
  PPot = (HP[behind][ahead] + HP[behind][tied]/2 +
HP[tied][ahed]/2) / (HPTotal[behind]+HPTotal[tied]/2)
  NPot = (HP[ahead][behind] + HP[tied][behind]/2 +
HP[ahead][tied]/2) / (HPTotal[ahead]+HPTotal[tied]/2)

  return (PPot, NPot)
}
```

The main advantage of this formula is that it considers future rounds, which is important because some games might reach showdown, being for this reason more precise than hand strength. Regarding disadvantages, this formula can't be used in River round (because the hand can't evolve no more). This formula cannot be used in Pre Flop rounds, because we can't calculate the hand strength for a two hand card. This might be solved by combining this algorithm with Chen Formula which will be explained later.

3.3.1 Combining hand potential with opponent modeling

Just like the Hand strength algorithm, if the Hand Potential is modified to only iterate over cards that the opponents might have [8], it is possible to obtain a better estimate over the chances of winning.

3.4 Effective Hand Strength

It is possible to calculate the probability of winning by combining the Hand Strength and Hand Potential algorithms (equation 2).

$$\mathbf{Pr(win)} = \mathbf{HS} \times (\mathbf{1} - \mathbf{NPot}) + (\mathbf{1} - \mathbf{HS}) \times \mathbf{PPot} \quad \textbf{(2)}$$

By setting the Negative Pot Potential to 0, it is possible to determine the effective hand strength which is the probability of the hand is either the best or will improve to become the best (equation 3).

$$\mathbf{EHS} = \mathbf{HS} + (\mathbf{1} - \mathbf{HS}) \times \mathbf{PPot} \quad \textbf{(3)}$$

3.5 Chen Formula

Chen formula is a mathematical formula developed by the professional poker player William Chen[9]. This formula can determine the relative value of the pocket hand. The main advantage of this formula over hand strength and current hand odds is that it does not need to generate permutations of card sets. For this reason, this algorithm is much faster than the others. The disadvantage of this algorithm over the others is that it only supports two card hands thus working only in Pre Flop rounds.

The following code represents the Chen Formula algorithm.

```
function ChenFormula(card1, card2){
  baseScore = Max(Score(card1), Score(card2))
  if(card1.value == card2.value) { /* if is pair */
    baseScore = Max(5, baseScore * 2)
  }
  if(card1.suit == card2.suit) {
    baseScore += 2
  }
  gap = Abs(card1.value - card2.value)
  switch(gap) {
    case 0: break;
    case 1: baseScore++; break;
    case 2: baseScore--; break;
    case 3: baseScore-= 2; break;
    case 4: baseScore-= 4; break;
    default: baseScore-= 5; break;
  }
  return baseScore - gap;
}

function Score(card) {
  switch(card.value) {
    case Ace: return 10; break;
    case King: return 8; break;
    case Queen: return 7; break;
```

```
      case Jack: return 6; break;
      default: return card.value / 2.0; break;
    }
  }
```

The algorithm is composed by two functions. The Score function returns a real number that scores a card. For instance, any ace card has the biggest score possible (10). The ChenFormula function returns an integer that represents the value of the hand. The max value of the returned value is 20 for a double Ace hand.

4 Hand Rank Evaluation

A poker hand rank evaluator is a function that receives a set of cards (5 to 7) and returns a number that means the relative value of that hand. This function is used to calculate lower level probabilities. For instance, every hand odds function described in this article (on the previous section) uses a rank function in order to determine if the player's hand is better than the opponent's hand.

Building an algorithm to determine the hand's rank is a relatively easy task, using naive methods, i.e. using an algorithm that intuitively makes sense [10]. Naïve hand rank evaluators usually follow the next steps:

- Sort the hand by card value (Deuce has the lowest value and Ace has the highest);
- Iterate across the hand, collecting information about ranks and suits of the cards;
- Make specific tests to check if the hand is of certain type (Flush, Straight, Three of a Kind, …);

The main problem of naïve evaluators is that they are slow. The hand odds evaluation functions call hand rank functions multiple times for each call, which means that having a slow hand rank function might slow down the calculation of the probability of success. The solution to this problem is to use top-down dynamic programming algorithms in order to speed up the rank function. The following subsections will present some hand rank functions.

4.1 Cactus Kev's 5-Card Evaluator

The Cactus Kev's 5-Card Evaluator uses one of fastest hand evaluator algorithms. The idea behind the algorithm is using a pre-computed hand ranking table. The number of possible combinations can be calculated as in equation 3.

$$N = \frac{(n!)}{(r!(n-r)!)}, n = 52, r = 5 \rightarrow N = \frac{52!}{5!(47!)} \rightarrow N = 2598960 \quad (3)$$

Since the number of combinations is not that high, it is quite computationally feasible to store all hands value in a 10mb table (2598960 * 4 bytes). The problem is that all hands must be in order. Cactus Kev's 5 card evaluator uses prime numbers to order the cards, because multiplying prime values of the rank of each card in your hand will result in an unique product, regardless of the order of the five cards [11].

Cactus Kev's evaluates cards with the following prime values: Two – 2; Three – 3; Four – 5; Five – 7; Six – 11; Seven – 13; Eight – 17; Nine – 19; Ten – 23; Jack – 29; Queen – 31; King – 37; Ace – 41.

For instance a King High Straight hand will always generate a product value of 14,535,931. Since multiplication is one of the fastest calculations a computer can make, we have shaved hundreds of milliseconds off our time had we been forced to sort each hand before evaluation [11].

The only big limitation of this hand evaluator is that it can only be used to evaluate 5-card hands. This means that to use it in game variations like Texas Hold'em which needs to evaluate 7-card hands (in River round), the function had to evaluate all possible 21 combinations of 5 cards to determine which one was the best.

4.2 Pokersource Poker-Eval

Poker-eval is a C library to evaluate poker hands. The result of the evaluation for a given hand is a number. The general idea is that if the evaluation of the player's hand is lower than the evaluation of the hand of its opponent, then it loses. Many poker variants are supported (draw, Hold'em, Omaha, etc.) and more can be added. Poker-eval is designed for speed so that it can be used within poker simulation software using either exhaustive exploration or Monte Carlo [10, 12]. Poker-eval is probably the most used hand evaluator, because of its multi-portability of poker variants and its speed of evaluation. The only limitation might be the complexity of its low level API, however there are some third party classes that encapsulate the usage of Poker-Eval API, making it simpler to use.

4.3 Paul Senzee's 7-Card Evaluator

Paul Senzee's 7 Card Evaluator [10, 13] uses a pre computed hand table to quickly determine the integer value of a given 7 card hand. Each hand is represented by a 52 bit number, where each bit represents an activated card. The total number of activated bits is 7, representing a 7 card hand.

If unlimited memory was available, it would be possible to index the resulting rank value into an enormous and very sparse array. However, this would require an array with 2^52 entries which means it would not be, nowadays, computationally feasible as it would require about 9 petabytes of memory (9 million gigabytes).

To solve the problem, Paul Senzee's developed a hash function that turns the hand value into an index between 0 and roughly 133 million and computed a 266mb which is by far a much smaller table, by grouping similar hand ranks.

The limitation of this hand evaluator is that it only evaluates 7 card poker hands, therefore is not portable to other poker variants.

4.4 TwoPlusTwo Evaluator

TwoPlusTwo evaluator is another lookup table poker hand evaluator with the size of 32487834 entries with a total size of ~250mb[10]. However TwoPlusTwo Evaluator is extremely fast, probably the fastest hand evaluator there is. To get the value of a given hand, the process is just performing one lookup per card. For instance to get a 7 card hand value the code with be just (admitting HR is the lookup table):

```
function Rank(cards) {
  p = HR[53 + cards[0]]
  p = HR[p + cards[1]]
  p = HR[p + cards[2]]
  p = HR[p + cards[3]]
  p = HR[p + cards[4]]
  P = HR[p + cards[5]]
  P = HR[p + cards[6]]
  return p
}
```

The idea behind the implementation of this extremely fast evaluator is a state machine. Each entry on the table represents a state. The next state is the sum of the value of the card and the value of the state. In the final state, the value represents the hand value.

This hand evaluator has no real limitations since it supports 5, 6 or 7 card hands.

4.5 Hand rank evaluators benchmark

In order to determine the fastest hand rank evaluator, a benchmark was performed. The results are shown in the following table.

Table 1. Hand rank function benchmark

Hand rank function	Elapsed time for 100.000 trials (ms)
Cactus Kev's 5-Card Evaluator	420 ms
Pokersource Poker-Eval	300 ms
Paul Senzee's 7-Card Evaluator	380 ms
TwoPlusTwo Evaluator	85 ms

As it can be observed, TwoPlusTwo Evaluator is by far the fastest hand rank evaluator.

5 Conclusions

There is still a long way to go to create an agent that plays poker at the level of the best human players. This research presented how an agent can measure the quality of its hand in order to aid its decisions during the game. This is rather important to build a competitive artificial agent in the future, because it is impossible to play well without knowing how strong the hand is.

There are five main hand odds evaluators to check the probability of winning taking in to account the current game state. Every one of them provides relevant information in each game round. Hand potential and hand strength algorithms were also modified in order to integrate opponent modeling with this measures.

With respect to hand rank evaluators, some of them proved to be very fast, which is important in because they are called many times by the hand odds algorithms. The fastest evaluator was TwoPlusTwo evaluator, achieving the smallest elapsed time in a simple experience.

In future work, the researchers should focus on combining even more this measures with opponent modeling techniques, in order to create better strategies.

References

[1] D. Billings, *et al.*, "Opponent modeling in poker," presented at the Proceedings of the fifteenth national/tenth conference on Artificial intelligence/Innovative applications of artificial intelligence, Madison, Wisconsin, United States, 1998.

[2] A. Davidson, "Opponent modeling in poker.," Master, Department of Computing Science, University of Alberta, 2002.

[3] D. Sklansky, *The Theory of Poker: A Professional Poker Player Teaches You How to Think Like One*: Two Plus Two, 2002.

[4] N. M. S. Ulam, "The Monte Carlo Method," *Journal of the American Statistical Association,* vol. 44, pp. 335-341 1949.

[5] P. Programmer. (2006, 02-01-2011). *Poker for Programmers: Poker Algorithms and Tools for the C# Programmer.* Available: http://pokerforprogrammers.blogspot.com/

[6] D. Billigs, "Algorithms and Assessment in Computer Poker," Doctor PhD, Department of Computing Science, University of Alberta, 2006.

[7] D. Billings, *et al.*, "The challenge of poker," *Artif. Intell.,* vol. 134, pp. 201-240, 2002.

[8] D. Felix, *et al.*, "An Experimental Approach to Online Opponent Modeling in Texas Hold'em Poker," presented at the Proceedings of the 19th Brazilian

Symposium on Artificial Intelligence: Advances in Artificial Intelligence, Savador, Brazil, 2008.

[9] B. C. a. J. Ankenman, *The Mathematics of Poker*, 1 ed.: Conjelco, 2009.

[10] (28 November). *Coding the Whell: Poker Hand Evaluator Roundup*. Available: http://www.codingthewheel.com/archives/poker-hand-evaluator-roundup

[11] (2006, 01/01/2011). *Cactus Kev's Poker Hand Evaluator*. Available: http://www.suffecool.net/poker/evaluator.html

[12] (2006-2010, 01-01-2011). *Pokersource Poker-Eval*. Available: http://pokersource.sourceforge.net/

[13] (2007, 01-01-2011). *Paul Senzee on Software, Game Development and Technology*. Available: http://www.senzee5.com/2007/01/7.html