Computing Card Probabilities in Texas Hold'em

Luís Filipe Teófilo[1,2], Luís Paulo Reis[1,3], Henrique Lopes Cardoso[1,2]
[1] LIACC – Artificial Intelligence and Computer Science Lab., University of Porto, Portugal
[2] FEUP – Faculty of Engineering, University of Porto – DEI, Portugal
[3] EEUM – School of Engineering, University of Minho – DSI, Portugal
luis.teofilo@fe.up.pt, lpreis@dsi.uminho.pt, hlc@fe.up.pt

Abstract— **Developing Poker agents that can compete at the level of a human expert can be a challenging endeavor, since agents' strategies must be capable of dealing with hidden information, deception and risk management. A way of addressing this issue is to model opponents' behavior in order to estimate their game plan and make decisions based on such estimations. In this paper, several hand evaluation and classification techniques are compared and conclusions on their respective applicability and scope are drawn. Also, we suggest improvements on current techniques through Monte Carlo sampling. The current methods to deal with risk management were found to be pertinent concerning the agent's decision-making process; nevertheless future integration of these methods with opponent modeling techniques can greatly improve overall Poker agents' performance.**

Keywords: Computer Poker; Poker hand probabilities; Opponent modeling; Texas Hold'em Poker; Incomplete information games; Game state abstraction

I. INTRODUCTION

Incomplete information games such as Poker became a field of interest for the AI research community over the last decade. This game presents unique challenges when compared to other strategy games like chess or checkers. In the latter, players are always aware of the full state of the game. On the other hand, Poker's game state includes hidden information, since each player can only see his/her cards and the community cards, making Poker a game which is much more difficult to analyze. Poker is also a stochastic game, i.e., it comprises the element of chance.

Due to the occurrence of random events (card dealing) and the existence of hidden information, the implementation of competitive Poker software agents greatly benefits from opponent modeling – categorize the opposing players in order to determine their possible strategies. When an agent identifies the opponents playing style, it can adapt its strategy to maximize the overall utility against that opponent. This is important as the goal of Poker is to score as much as possible and not to win a particular game (or a particular set of games).

This paper describes several techniques on how a Poker agent can measure the risk of its actions, with emphasis on the available information: the pocket cards and the community cards. An agent can benefit of accurately knowing how strong its cards are. This allows agents to adapt their decision stream to better manage their resources.

The rest of the paper is organized as follows. Section 2 describes the game which is the domain of this paper: the Texas Hold'em variant of Poker. Section 3 describes current methods and tools to quickly compute the rank of a hand, which is essential for probabilities calculation. Section 4 describes the current techniques to compute the odds of a 5 to 7 card hand as well as an explanation on how to improve the efficiency of these techniques. Section 5 describes how to estimate the value of pocket cards, i.e., when the community cards are still hidden. Finally, the main conclusions and some suggestions for future work are drawn in Section 6.

II. BACKGROUND

Poker is a class of card betting games with similar rules. This work focus on a particular variant of Poker – Texas Hold'em – which is the most popular set of rules nowadays. Hold'em rules also have specific characteristics that allow for new developed approaches to be adapted to other variants with reduced effort [1].

The game is based upon the concept of players betting that their current hand [1] is stronger than the hands of their opponents. All bets throughout the game are placed in the pot and at the end of the game the player with the highest ranked hand wins the game and collects the pot. Alternatively, it is also possible to win the game by inducing opponents to fold their hands by making bets that they are not willing to match. Thus since players' cards (pocket cards) are hidden, it is possible to win the game with a hand with lower score. This is done by convincing the opponents that one's hand is the strongest one. This particular feature of the game's rules makes it even more difficult to develop an agent's strategy, because the agent cannot assess if it has made a good decision.

A. Rules

Texas Hold'em Poker is a community card[2] Poker variant. In each game there is a minimum bet and at the start two cards are dealt to each player – pocket cards. A dealer player is assigned and marked with a dealer button. The dealer position

[1] Hand – a set of cards of a particular player
[2] Community card – visible card that is shared by all players.

This work was financially supported by FCT – Fundação para a Ciência e a Tecnologia through the Ph.D. Scholarship with reference SFRH/BD/71598/2010.

rotates clockwise from game to game. The game starts by the player on the left of the dealer position posting the small blind (half of the minimum bet) and the player on his left posting the big blind (minimum bet). The first player to act is the one on the left of the player posting the big blind.

The game is composed of four rounds: Pre-Flop, Flop, Turn and River. The participants play subsequently in turns and they can choose to match the highest bet (Call), increase that bet (Raise) or forfeit the game and lose the pot (Fold). A player wins if he/she is the last standing player or if he/she has the highest ranked card after the last round at the Showdown. This Poker variant has two sub-variants with a small difference in their rule set. These are called Limit and No Limit Texas Hold'em. The difference between them is the existence of a raise value limit.

B. Ranking hands in Poker

A Poker hand is a set of five cards that expresses the player's score. Being Δ the set of all cards in the deck, Φ the set of pocket cards of a particular player and Ω the set of community cards so that $\Phi \cap \Omega = \varnothing$. Thus, the score function is defined as $s: [\Delta]^5 \rightarrow \mathbb{N}$. For a particular player, the hand h is the union of the pocket cards and the community cards ($\Phi \cup \Omega$). Thus, the player's score is given by the rank function, as follows (equation 1):

$$Rank(h) = \max(\{\forall\, x \in [\Phi \cup \Omega]^5 : s(x)\}) \quad (1)$$

The possible hand ranks are (from stronger to weaker): Straight Flush (sequence of same suit), Four of a Kind (4 cards with same rank), Full House (Three of a Kind + Pair), Flush (5 cards with same suit), Straight (sequence), Three of a Kind (3 cards with same rank), Two Pair, One Pair (2 cards with same rank) and Highest Card (not qualifying to other ranks). Examples of each rank are demonstrated in table I. These ranks are not equally valued. Each rank has sub-ranks essentially based on the score of the top cards (e.g.: a pair of aces scores higher than a pair of queens). In total, there are 7462 possible sub-ranks in Texas Hold'em Poker.

TABLE I. POKER HAND RANKS WITH EXAMPLES.

Hand Name	Example of card set					#distinct sub-ranks
Straight Flush	8♠	7♠	6♠	5♠	4♠	10
Four of a Kind	A♣	A♦	A♥	A♠	K♠	156
Full House	Q♣	Q♠	7♥	7♠	7♦	156
Flush	T♥	8♥	6♥	4♥	2♥	1277
Straight	4♦	5♥	6♦	7♠	8♠	10
Three of a Kind	T♣	T♦	T♥	Q♣	3♦	858
Two pair	7♣	7♠	3♠	3♥	Q♠	858
One pair	2♠	2♣	8♣	7♣	3♥	2860
High Card	A♥	T♥	6♦	4♣	2♣	1277
					Total:	7462

It is important to distinguish the concepts of rank and odds. The rank is the final score of a hand. This means that at the end of the game, the player with the better rank will win the pot. The odds indicate the probability of a certain hand being the best at the end of the game. It consists of computing several hand ranks and the average rank of several possible opposing hands. In other words, they specify how likely it is for a particular player to win the pot at the Showdown.

C. Opponent Modeling in Texas Hold'em

Since Texas Hold'em Poker is an incomplete information game, for competent play, an agent should be able to model the opponents' strategies in order to predict the opponents' actions and also to predict the possible outcomes of their own choices. By predicting the opponents' actions, the agents will be more likely to optimize their revenue and their expected utility over time.

One good example of opponent modeling are the Sklansky groups [3]. The Sklansky groups are a card abstraction technique – grouping different card sets and playing with an equal strategy for each group. This abstraction clusters all combinations of pocket cards into eight groups and defines how frequent each group is for each player type, based on the concept of aggression factor[3] and VP$IP[4].

Opponent models can also be automatically computed. Popular techniques are for instance machine learning [2–4], reinforcement learning [5] or Bayesian networks [6].

III. HAND RANK COMPUTATION

A poker hand rank evaluator is a software program that computes the value of the rank function (equation 1), partially computed by the score function $s: [\Delta]^5 \rightarrow \mathbb{N}$. In Texas Hold'em Poker this evaluator receives as parameter the set of cards $\Phi + \Omega$, where $\#\Phi = 2 \wedge \#\Omega \in \{5,6,7\}$. The evaluator returns a natural number representing the relative value of that hand (typically from 0 to 7461, where 7461 corresponds to the one of the top scored Straight Flushes).

To compute the probability of success of a given hand – odds – it is usually necessary to compute several hand ranks before. For instance, the methodologies described in Sections IV or V require the computation of hand ranks.

Programming an algorithm to determine the hand's rank is a relatively trivial task. This can be done using a naïve approach, i.e. using an algorithm that intuitively makes sense and that is humanly readable. Naive hand rank evaluators usually consist of the following steps:

- Sort the hand by card value (deuce has the lowest value and ace has the highest);
- Iterate through the hand, collecting information about ranks and suits of the cards;
- Make specific tests to check, iteratively, if the hand is of a certain rank, starting at the higher ranks.

One example to illustrate this idea can be found in the code bellow. This example does not consider the whole set of Texas Hold'em rules (the sub-ranks referred on section II.B).

```
Function HandRank(Hand) {
  Sort(Hand);
  If IsStraightFlush(Hand) Return 9;
  If IsIsFourOfAKind(Hand) Return 8;
  If IsFullHouse(Hand) Return 7;
```

[3] Aggression factor – number of calls divided by the number of raises
[4] VP$IP – voluntary put money in the pot before the Flop round

```
  If IsFlush(Hand) Return 6;
  If IsStraight(Hand) Return 5;
  If IsThreeOfAKind(Hand) Return 4;
  If IsTwoPairs(Hand) Return 3;
  If IsOnePair(Hand) Return 2;
  Return 1;
}
```

The problem with naïve evaluators resides in their efficiency, which is important because the rank evaluator is used by a hand odds evaluator several times per computation. The solution to this problem resides in top-down dynamic programming algorithms in order to speed up the rank function. The next subsections will present some developed approaches to solve this issue.

A. Pokersource Poker-Eval

Poker-Eval is a C implementation of a Poker Hand rank evaluator [7]. As described at the beginning of this section, given a hand, this evaluator returns a natural number that represents the hand score. This evaluator uses a naïve approach and, to the best of our knowledge, the fastest one.

The main advantages of this evaluator is its architecture which supports multi Poker variants, multi-platform usage since there are wrappers for other programming languages and its low memory usage when compared to look-up table based approaches. The main issue of this evaluator is its low level API which makes it harder to use by programmers.

B. Cactus Kev

The Cactus Kev's 5-Card Evaluator [8] is a system to compute 5 card hand rank. The idea behind its algorithm is the construction of a pre-computed look-up table with every possible rank. However, since the number of possible 5 card sequences is ${}_{52}P_5$, the size of the table would be about 2.5 GB of memory (considering 8 bytes to store the hand and its rank). To solve this problem one can group similar hands (same cards, different order), resulting in $\binom{52}{5}$ hands, turning this solution feasible (the size of the new look-up table would be about 20 MB). However, this solution requires sorting the hand cards before accessing the look-up table, wasting additional CPU time. To solve this, Cactus uses a 32 bit integer representation of the cards (Figure 1).

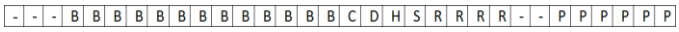

Figure 1. Cactus Kev's card representation

P (6 bits) represents the value of a card in a form of a prime number, with the following 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. The reason behind this decision resides in the fact that the multiplication of two prime numbers always generates a unique value. This allows for avoiding the step of sorting the hand cards, saving CPU time. Therefore, the product of these values of card values can be used to index the hands.

R (4 bits) represents the rank of the card (Two – 0; Three – 1; Four – 2; …). **CDHS** represents the card's suit mask, where one of the bits is activated (C if the card is Clubs, D if the card is diamonds …). The **B** (12 bits) represents the card's rank mask, where the first bit is activated when the card is a Two or the second bit is activated when the card is a Three, and so on.

Three lock-up tables are used in this evaluator: **flushes** (the ranks of all flushes and straight flushes hands), **unique5** (the ranks of all hands with cards with different ranks) and **values** (the remaining cards). To build the look-up tables, a naïve evaluator is required.

To find the value of a certain hand, the three tables are consulted. Assuming the cards of the hand are labeled as C_1, C_2, C_3, C_4 and C_5, Cactus first verifies if the hand is a flush:

```
Index = C1 AND C2 AND C3 AND C4 AND C5 AND 0xF000
```

For the calculated index, the table can either return the value of the hand or 0, if the hand is not a flush or a straight flush. The next step is to verify if the hand belongs to **unique5** by calculating the index the following way:

```
Index = (C1 OR C2 OR C3 OR C4 OR C5) >> 16
```

Once again, if the value of the table at the calculated index is 0, we have to look for the result in another table. The final index (equation 2) uses the described prime number strategy.

$$Index = \prod_{i=1}^{5} (C_i \; AND \; 0xFF) \tag{2}$$

The problem of using this index system is that it would result in a very large look-up table of size $41 \times 41 \times 41 \times 41 \times 37 = 104{,}553{,}157$. The author of this technique solves this problem by storing the indexes in a binary search tree for fast hand value retrieval.

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

C. Paul Senzee

Paul Senzee's hand evaluator is an improved version of Cactus Kev. However, instead of using a binary search, Senzee uses a pre-computed perfect hash table.

A perfect hash table guarantees no collisions in the storage of the hands' values. Also it allows for acquiring the values in constant time instead of the $O(\log n)$ complexity of the binary search. The used hash function was based on [9]. This approach produced a time improvement factor of about 2.7 times [10].

Another advantage of Paul Senzee's approach it that there is a version of the evaluator for 7 cards, which is useful for the river round (instead of computing 21 ranks).

Paul Senzee's 7 Card Evaluator also uses a pre computed hand table to quickly determine the integer value of a given 7 card hand. For 7 hand cards lookup, Paul represents each hand with a 52 bit string, 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 with 2^{52} entries of about 9 petabytes of memory (9 million gigabytes) assuming two bytes per each entry (short integer). To solve this problem, Paul Senzee's developed a hash function that turns the hand value into an index between 0 and roughly 133 million and, by using the Cactus Kev's evaluator, it is possible to produce a 266MB lookup table. The author of this approach does not provide information about the hash generation code. The main limitation of the 7 card version of Paul's evaluator is that it only supports 7 cards (it does not support Flop and Turn rounds).

D. TwoPlusTwo Evaluator

TwoPlusTwo evaluator is a lookup table Poker hand evaluator that uses a table of 32,487,834 entries with a total size of ~130 MB [11]. The TwoPlusTwo Evaluator is extremely fast and probably the fastest hand evaluator there is. This is because the ranks of the hands being stored in a non-sparse array without redundant values. This means that each position at the lookup table represents a unique hand (the size of the table is exactly $\binom{52}{7}$, the number of possible 7 card hands). Using this structure, to get the value of a given hand, only one lookup per card is performed. For instance, the following function will compute a 7 card hand value, being HR the lookup table.

```
Function Rank(Hand) {
 Return HR[HR[HR[HR[HR[HR[53 + Hand[0]] +
Hand[1]] + Hand[2]] + Hand[3]] + Hand[4]] + Hand[5]]
+ Hand[6]] }
```

To store the hands, the implementation of this evaluator is based on a state machine. Each entry in the table represents a state. The next state accumulates in the current state and the value of the card. In the final state, the value represents the hand value. This hand evaluator supports 5, 6 or 7 card hands.

E. Hand rank evaluators benchmark

In order to determine the fastest hand rank evaluator, a benchmark test was performed. The test consisted of ranking a pre-computed sequence of all possible combinations of 5 card hands (2,598,960 hands). The tests were performed 1000 times each on an Intel I7-3940XM CPU. The set of hands was tested with each described hand rank evaluators and the results are presented in Table II.

TABLE II. HAND RANK FUNCTION BENCHMARK

Hand rank program	**Average elapsed time for 1.000 trials (ms)**	
	Non parallel	Parallel (4 cores)
Cactus Kev	807,13	591,22
Pokersource	2.520,44	980,14
Paul Senzee	403,04	195,44
TwoPlusTwo	91,09	37,98

The TwoPlusTwo Evaluator is by far the fastest hand rank evaluator, performing better in both experiments.

IV. COMPUTING ODDS ON COMPLETE HANDS

Evaluating the odds of a hand consists of measuring the quality of a given hand in any game stage. This section describes how to compute the probability of a certain hand being successful at Showdown (last round where the winner is decided). By evaluating the hand it is possible to determine the probability of winning or losing the current game. This knowledge can be used to inform the agent's decision of either fold the hand or play it, as well as to assess the probability of success and the risk that the agent is facing. Computing the hand odds may consider the following variables: Pocket cards; Number of opponents; Community cards; Possible community cards to come; Possible opponents' cards.

The hand evaluation method typically returns a real number between 0.0 and 1.0. If it returns the lower limit, this means that the hand will lose regardless of future events in the game, unless the player uses deception to bring opponents to forfeit. Conversely, obtaining the upper limit from the hand evaluation function means that victory is mathematically assured – the only way of losing is to unwisely fold the hand.

A. Hand Strength

The hand strength [1] is the probability of the current hand being the best if the game reaches a showdown with all remaining players. It consists of enumerating all possible hands that an opponent can have and checking if the agent's hand is better than the hands in the enumeration. By counting the number of times the player's hand is found to be better, it is possible to measure the quality of the hand. Using subsection II.B terminology, the hand strength (*HS*) for a given number of opponents *n* is given by:

$$Rem = [\Delta \backslash \Phi]^5$$

$$Ahead(h) = \#\{\forall x \in \text{Rem}: s(x) > \text{Rank}(h)\}$$

$$Tied(h) = \#\{\forall x \in \text{Rem}: s(x) = \text{Rank}(h)\}$$

$$Behind(h) = \#\{\forall x \in \text{Rem}: s(x) < \text{Rank}(h)\}$$

$$HS_n(h) = \left(\frac{Ahead(h) + \frac{Tied(h)}{2}}{Ahead(h) + Tied(h) + Behind(h)} \right)^n$$

The Hand Strength may be used in any round of the game. However hand strength does not address the possibility of the hand improving in subsequent rounds of the game, which is possible because in Texas Hold'em new cards are revealed at the start of every round (community cards). This issue is addressed by the Hand Potential Formula [1] which sums up possible hand strengths in subsequent rounds (described in the next subsection).

In [12], the authors suggest 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 this purpose, the proposed algorithm would use $Remain' =$

$[\Delta\backslash\Phi\backslash\partial]^5$ where ∂ is the set of cards that the opponent probably hasn't given that $\Phi \cup \Omega \cup \partial = \Delta$, and $\Phi \cap \Omega \cap \partial = \varnothing$. This approach was successfully tested in Texas Hold'em heads up[5] games.

B. Hand Potential

Hand potential [1], [12] 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 deck cards to be revealed. This means that the current hand rank might improve, since the hand is composed of the set of five available cards (pocket or community cards) that has the highest rank among all available cards.

This algorithm is an extension of 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' hands 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 finally wins are calculated.
- Negative potential: of all possible games with the current hand, all the scenarios where the agent is ahead but finally loses are calculated.

The components of hand potential can be calculated as follows (again using subsection II.B terminology):

$$Rem = [\Delta\backslash\Phi]^5,\ RemB = [\Delta\backslash\Phi]^{\text{round}}$$

$$PPOT_n = \#\{HS_n(x) \geq HS_n(x+y) \colon \text{x} \in \text{Rem} \wedge \text{y} \in RemB\}$$

$$NPOT_n = \#\{HS_n(x) \leq HS_n(x+y) \colon \text{x} \in \text{Rem} \wedge \text{y} \in RemB\}$$

where round is 2 when performing computation for the Flop round and 1 for the Turn round. To better illustrate this approach, we present the following pseudo-code that represents the hand potential algorithm.

```
function HandPotential(ourcards, boardcards){
  int array HP[3][3], HPTotal[3] /* Init to 0 */
  ourrank = Rank(ourcards, boardcards)
  for each case(oppcards) {
    opprank = Rank(oppcards, boardcards)
    if(ourrank>opprank) index = ahead
      else if(ourrank=opprank) index = tied
        else index = behind
    HPTotal[index]++
    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 calculation is the consideration of Texas Hold'em upcoming rounds. This is important because some games might reach showdown, therefore presenting a more refined measure than hand strength. This algorithm presents the same result as Hand Strength in the River round (because the hand cannot evolve any further). Moreover, this method cannot be used in Pre Flop rounds, because it is not possible to calculate the hand strength for a two hand card. This might be solved by combining this algorithm with Chen Formula (see Section V). Similarly to the hand strength algorithm, if the Hand Potential is modified to only iterate over cards that the opponents might have [12], it is possible to obtain a better estimate of the winning ratio.

C. Effective Hand Strength

The probability of winning can be calculated by combining the Hand Strength with the Hand Potential components.

$$\Pr_n(\text{win}) = \text{HS}_n \times (1 - \text{NPot}_n) + (1 - \text{HS}_n) \times \text{PPot}_n$$

By setting the NPOT to 0, it is possible to determine the effective hand strength which is the probability of the hand either being the best or improving to become the best.

$$\text{EHS}_n = \text{HS}_n + (1 - \text{HS}_n) \times \text{PPot}_n$$

D. Monte Carlo Effective Hand Strength

The effective hand strength algorithm is heavy in the number of cycles, especially due to the use of hand potential algorithm, because we have to generate all possible sequences of board cards to come. For instance, if we are competing against 2 players and we are in the Flop round (3 community cards show), to compute the hand potential we have to generate $P_{n,k} = P_{52-3-2,3+2+2} = P_{47,7} \approx 3.17 \times 10^{11}$ permutations, which is (in most cases) unfeasible. To solve this, we introduce the usage of Monte Carlo Method. This consists of sampling a fixed number of random possible board cards. Table III demonstrates the errors obtained by randomly sampling a certain number of board cards. Our experiments show that sampling 100 board cards for each algorithm step produces an already negligible error.

TABLE III. SAMPLING BOARD CARDS

Number of Samples	Number of iterations	Error
All samples	$\approx 3.17 \times 10^{11}$	0
1000	$10^4 \times P_{45,4}$	~0.001
100	$10^3 \times P_{45,4}$	~0.012
10	$10^2 \times P_{45,4}$	~0.151

[5] Heads up – Poker game between two players.

V. COMPUTING ODDS ON INCOMPLETE HANDS

The previous section has shown how to compute the odds of complete hands i.e. when it is possible to compute the hand rank. In this section the Pre-Flop round of the game is addressed by showing how to compute odds when all community cards are hidden.

A. Chen Algorithm

Chen method is a fast naïve algorithm developed by the professional poker player William Chen [13]. This formula can determine the relative value of the pocket hand. The main advantage of this is that it does not need to generate permutations of card sets. For this reason, this algorithm is much faster than all others based on Hand Strength.

```
Function Chen(card1, card2)
   score = Max(Score(card1), Score(card2))
   if(card1.suit == card2.suit)
      score = score + 2
   switch(Abs(card1.Rank - card2.Rank))
      case 0: score = score * 2
      case 1: score = score + 1
      case 2: score = score - 1
      case 3: score = score - 2
      case 4: score = score - 4
      default: score = score - 5
   return score
```

The algorithm is composed of two functions. The Score function returns a real number that scores a card. (Ace – 10, King – 8, Queen – 7, Jack – 6 and Value / 2 for others). The Chen Formula function returns an integer which represents the value of the hand. Thus, the maximum value of the returned value is 20 for a double Ace hand. In order to maintain consistency among odds evaluation methods, we introduce output normalization. The normalized Chen formula is:

$$\text{NormalizedChen}(c_1, c_2) = \frac{Chen(c_1, c_2) - \min(\{Chen(c_1, c_2): c_1 \in \Delta \wedge c_2 \in \Delta\})}{\max(\{Chen(c_1, c_2): c_1 \in \Delta \wedge c_2 \in \Delta\})}$$

B. Computing the strength of two hand cards

One possible approach to calculate the hand probabilities for incomplete hands is to combine the effective hand strength formula with Monte Carlo sampling. This process consists of constructing a pre-computed table by simulating enough games (with [14]) for each possible pocket hand and computing the average number of wins and losses for that hand. The agents used in this simulation used a random fixed mixed strategy with equal probabilities for call, raise or fold. After obtaining the simulation values, the game's isomorphisms were discarded (e.g. 4♣, 3♠ should have the same value as 4♥, 3♦) by calculating their average value. Finally all hands were sorted and saved in a pre-computed table. The use of a pre-computed table is justified since computing the hand relative values takes a long time. Moreover, the size of the produced table is short: $P_{n,k} = P_{52,2} = \frac{52!}{(52-2)!} = 2652$ entrances.

VI. CONCLUSIONS

There is still work to be done in order to create an agent which can play Poker at the level of the best human players. This research has shown how an agent can assess the quality of its hand so as to aid its decision-making process during the game. This is a rather important feature when developing a competitive artificial agent, since it is impossible to play well without estimating how strong the agent's hand is. There are five main hand odds evaluators to calculate the winning ratio, which take into account the current game state. All of them provide relevant information for each game round. Also, modifications to hand potential and hand strength algorithms have been suggested in order to integrate opponent modeling into these methodologies. With respect to hand rank evaluators, some of them proved to be very fast, which is a crucial feature, since they are executed many times throughout a game. The fastest evaluator was TwoPlusTwo, achieving the smallest elapsed time in a simple experiment. In future work, researchers should focus on combining these evaluators further with opponent modeling techniques.

REFERENCES

[1] D. Billings, A. Davidson, J. Schaeffer, and D. Szafron, "The challenge of poker," *Artificial Intelligence*, vol. 134, no. 1–2, pp. 201–240, 2002.

[2] L. F. Teófilo and L. P. Reis, "HoldemML: A framework to generate No Limit Hold'em Poker agents from human player strategies," in *6th Iberian Conference on Information Systems and Technologies (CISTI 2011)*, 2011, pp. 755–760.

[3] L. F. Teófilo and L. P. Reis, "Building a No Limit Texas Hold'em Poker Playing Agent based on Game Logs using Supervised Learning," in *Proceedings 2nd International Conference on Autonomous and Intelligent Systems*, 2011, pp. 73–83.

[4] G. Nicolai and R. J. Hilderman, *No-Limit Texas Hold'em Poker agents created with evolutionary neural networks*. Ieee, 2009, pp. 125–131.

[5] L. F. Teófilo, N. Passos, L. P. Reis, and H. L. Cardoso, "Adapting Strategies to Opponent Models in Incomplete Information Games: A Reinforcement Learning Approach for Poker," in *Autonomous and Intelligent Systems - Third International Conference (AIS2012)*, 2012, pp. 220–227.

[6] R. J. S. Baker, P. I. Cowling, T. W. G. Randall, and P. Jiang, *Can opponent models aid poker player evolution?* Ieee, 2008, pp. 23–30.

[7] "Pokersource Poker-Eval." [Online]. Available: http://pokersource.sourceforge.net/.

[8] C. Kev, "Cactus Kev's Poker Hand Evaluator." [Online]. Available: http://www.suffecool.net/poker/evaluator.html.

[9] E. A. Fox, L. S. Heath, Q. F. Chen, and A. M. Daoud, "Practical minimal perfect hash functions for large databases," *Commun. ACM*, vol. 35, no. 1, pp. 105–121, 1992.

[10] Senzee5, "PAUL SENZEE ON SOFTWARE, GAME DEVELOPMENT, TECHNOLOGY AND LIFE," 2006. [Online]. Available: http://www.paulsenzee.com/2006/06/some-perfect-hash.html.

[11] "Coding the Whell: Poker Hand Evaluator Roundup," 2008. [Online]. Available: http://www.codingthewheel.com/archives/poker-hand-evaluator-roundup.

[12] D. Félix and L. P. Reis, "An Experimental Approach to Online Opponent Modeling in Texas Hold'em Poker," in *SBIA '08 Proceedings of the 19th Brazilian Symposium on Artificial Intelligence: Advances in Artificial Intelligenc*, 2008, pp. 83 – 92.

[13] B. Chen and J. Ankenman, *The Mathematics of Poker*, 1st editio. Conjelco, 2006.

[14] L. F. Teófilo, R. Rossetti, L. P. Reis, and H. L. Cardoso, "Simulation and Performance Assessment of Poker Agents," in *Springer LNCS 7838 (MABS 2012)*, 2013, pp. 69–84.