Abstraction Methods for Game Theoretic Poker

Jiefu Shi[1] and Michael L. Littman[2]

[1] Department of Computer Science
Duke University
Durham, NC 27708
jshi@cs.duke.edu
[2] AT&T Labs–Research
180 Park Ave. Room A275
Florham Park, NJ 07932-0971 USA
mlittman@research.att.com

Abstract. Abstraction is a method often applied to keep the combinatorial explosion under control and to solve problems of large complexity. Our work focuses on applying abstraction to solve large stochastic imperfect-information games, specifically variants of poker. We examine several different medium-size poker variants and give encouraging results for abstraction-based methods on these games.
Keywords: poker, game theory, imperfect information games, Texas Hold'em.

1 Introduction

One of the principle challenges in developing computer-based solutions to real-world problems is dealing with uncertainty. Four principle types of uncertainty are:

- *effect uncertainty*: In an unfamiliar environment, a decision maker might not know the possible effects of its decisions. We do not treat this type of uncertainty in this paper, although work in the area of reinforcement learning [2, 6] has focused on this issue.
- *outcome uncertainty*: In games of chance, for example, although the decision maker may know the set of possible outcomes and their probabilities, there is no way to know exactly which outcome will occur. For example, dealing a random card from a deck could result in a red card or a black card being dealt, but we would not know which until the card is shown.
- *state uncertainty*: The decision maker might not know information that may affect the outcomes of its future decisions. For example, in the game of Scrabble$^{\text{TM}}$, a player cannot see the other player's tiles.
- *opponent uncertainty*: In multi-agent systems, and especially games, the decision maker does not know precisely how other agents in the system will respond.

Traditional work in game theory has a mathematical framework for reasoning about uncertainty. Outcome and state uncertainty are modeled by probability theory; decision makers maintain probability distributions over the current state and future events, and decisions are made to maximize expected utility. Because opponents need not behave according to any fixed probability distribution, however, opponent uncertainty is handled differently. In the *game-theoretic approach* [7], an agent makes decisions to maximize its expected utility assuming the opponent makes decisions to minimize the agent's utility. Thus, although the decision maker does not know how the opponent will actually behave, it does as well as possible in the worst case.

The game-theoretic approach has been applied to a large number of games, such as chess, backgammon, checkers, and many others. In fact, it is safe to say that this is the predominant approach in use in computer games research. Billings et al. [1] argue that this approach is not appropriate for games such as poker where (a) opponent play is probably not very strong, and (b) repeated encounters give the decision maker an opportunity to learn patterns in the opponent's play. Thus, opponent uncertainty can, and perhaps should, be treated as a type of state/effect uncertainty.

We have decided to attack poker using a game-theoretic approach, mainly because we believe this is still the best way to create an extremely high-quality player. Learning techniques run the risk of being fooled into low-quality play by a sufficiently clever adversary and the game-theoretic approach guards against this. In addition, finding an optimal or approximately optimal strategy using game theory can lead to unexpected insights into the structure of the game. We describe an instance of this from our poker player in Section 5. Note that, throughout the paper, we treat only 2-player zero-sum poker.

In Section 2 we introduce the game of poker and some of its unique challenges. Sections 3 and 4 describe two techniques we developed for attacking poker games using game theory. Section 5 presents some results and Section 6 concludes.

2 Poker Introduction

Consider the following mini-poker game. We start with a deck of 3 cards: J, Q, K. We deal one card to each of two players. Each player contributes one dollar to the pot (ante) and looks at his or her card. Next, a betting round commences. In the betting round, the players look at their cards, then alternate either betting (adding a dollar to the pot) or passing. If a player passes when the other player has contributed more money to the pot, that player "folds" and forfeits the pot. On the other hand, if there are two consecutive passes or both players have added the maximum number of dollars to the pot (1, for this example), the players reveal their cards and the one with the higher card wins the pot.

We seek a game-theoretic strategy for this game. Consider the game tree shown in Figure 1, where the leaves are labeled with player 1's winnings if the corresponding branches are followed. If the game were one of perfect information (players can see each others' card), an optimal strategy could be found by "min-

imaxing" up from the leaves. This computes, for each node in the tree, what player 1 would win on average if she bet optimally while player 2 made optimal responses.

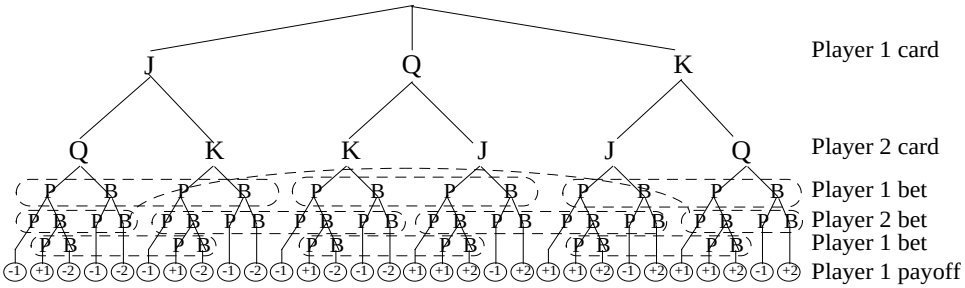

Fig. 1. Game tree for a very small poker game

Of course, a complete-information version of this game would be pointless, as whichever player was dealt the lower card would pass and the other player would win the ante. The challenge here is that, because cards are private, the game-tree nodes that share information sets with each other (ovals in the figure) are indistinguishable and the player must make the same choice from states in these sets. Using a simple bottom-up approach, it is not at all evident how this constraint can be satisfied. Further, optimal play in imperfect information games is often randomized and simple game-tree search cannot reveal this.

Koller et al. [3] provide an algorithm that maps a game tree notated with information sets to an optimization problem, namely a linear program. When this optimization problem is solved, it reveals the optimal (possibly randomized) strategy for the game. Roughly speaking, variables in the linear program represent stochastic decisions, constraints represent making the same decision from all states in the same information set, and the objective function captures the optimal expected minimax winnings.

An optimal strategy for this type of game can have very rich and interesting structure. Consider what player 1 could do when she has the J. Player 2 might have either the Q or the K. In either case, player 1 will lose, but if player 2 has the Q and player 1 can convince him she has the K (perhaps by betting whenever possible), player 2 might fold and she would win. This is an example of "bluffing". Consider also what player 1 could do if she had the K. If she bets grudgingly (only matching player 2's bets when necessary), she might be able to get player 2 to bet before beating him. This is sometimes called "slow playing". These strategies are practiced by experienced poker players and emerge from game-theoretic solutions as well [4].

Thus, interesting and surprising strategies for complex imperfect information games can emerge from game-theoretic solutions. Furthermore, using the linear-

programming approach, solution time is polynomial in the size of the game tree. Unfortunately, these game trees can get quite large. Consider a poker game in which c cards are dealt from a deck of d cards and players carry out b betting rounds each with a maximum of r raises per round.[1] The game tree for such a game contains at least $(d-c+1)^c\ 2^{rb}$ leaves. The justification for this equation is that the branching factor for each of the c dealt cards is the number of cards not yet dealt (at least $d-c+1$). Each of the b betting rounds consists of at most r raises, each of which can at most double the game-tree size (one for each order the two players carry out their raises).

Koller and Pfeffer [4] report solving games of size $(b,c,d,r) = (1,2,128,1)$ (more than 32K leaves), $(1,2,3,11)$ (more than 8K leaves), and $(1,10,11,3)$ (more than 8K leaves). Poker games played by people are much larger. The two-player version of Texas Hold'em, discussed in Section 4, has $(b,c,d,r) = (4,9,52,3)$ or more than 3.7×10^{18} leaves. Note that, often, a more appropriate representation for a game is a directed acyclic graph, since the order cards are dealt is not important sometimes (player's hole cards or the flop in Texas Hold'em). We do not consider this optimization in our analysis, although it can substantially decrease the game representation. In either case, though, full games are well beyond what can be solved using the linear-programming approach. In this paper, we examine several abstraction-based approaches that can be used to generate approximations to large imperfect information games. Note that a related theoretical treatment of game-theoretic abstraction is underway [5].

Section 3 describes binning methods, which effectively reduce the number of cards in the deck d. Section 4 describes ways of treating betting rounds independently to effectively reduce the number of betting rounds r. Section 5 describes the results of applying these ideas in a game we invented called Rhode Island Hold'em.

3 Binning

The simple three-card game in Section 2 can be easily scaled up by considering larger decks. Even with a 52-card deck and 3 raises, the game is not very challenging to solve (around 21K leaves in the game tree). However, it is not a huge leap to imagine a poker variation where players are dealt 5-card hands and must bet on them. This is similar to, but not identitical to, dealing each player one card from a deck of size 52 choose 5 (around 2.6M) cards with the highest card winning, since there is a well defined total order over all possible hands. The full game tree has more than 5.4×10^{13} leaves.

For games like this, with a single betting round based on the entire hand, we can use a grouping method to reduce the number of distinct hands considered. We rank all possible hands by their strength to obtain a ranking of each hand (a hand with higher value always beats a hand with lower value). For any hand,

[1] Throughout this paper, we use "raise" to mean putting money in the pot after the ante. This is different from the term "betting levels" used by some authors, which does not count the first bet as a raise.

we can determine its ranking very fast either by looking it up in a database, or just calculating it in real-time.

Next, we group hands into equal-size bins. Each bin contains hands with similar rankings. The game is then solved at the level of bins: we imagine that players are randomly assigned to bins, with the highest ranking bin the winner and ties broken arbitrarily; betting strategies are computed for the resulting game. The number of bins used in the approximation controls the degree of abstraction and can be adjusted to accommodate space and time requirements.

At a high level, this may not be too different from how humans play poker. We do not really care whether we have two kings with a five of spades or whether we have two kings with a six of diamonds; we treat these situations fairly similarly in terms of our decision making. Also, note that if we produce a strategy using the same number of bins as the total number of hands possible, then what we have is the true optimal strategy. Of course, the accuracy of the approximation depends on how the groups are formed.

For our test-bed, we used a game with 200 possible hands. Both players initially ante one chip into the pot. Each player is then dealt a hand and the betting round begins. For this game, $(b, c, d, r) = (1, 2, 200, 1)$. Its game tree contains at least 79K leaves.

We first generated the optimal strategy for player 1. We then ran experiments dividing hands into from 4 to 200 bins and produced strategies for player 2 based on each of these groupings. So, for example, when we used 50 bins to solve our game, bin number 1 (the lowest bin) contained the lowest hands of 1, 2, 3 and 4. Similarly, bin number 50 contained the highest hands of 197, 198, 199, and 200.

Figure 2 shows how well player 1 fared against the optimal strategy for player 2 based on 1,000,000 games (we felt this number of games would be sufficient to see statistically significant results). Note that player 2 has an advantage in this and most poker games by virtue of gaining information from seeing the other player's initial selection. The results are quite encouraging; using as few bins as 10% of the number of hands, the resulting play is almost as good as that of the optimal strategy.

In popular poker games, players must bet based on partial hands and the principle concern is "hand potential". The ranking trick just described does not directly apply here because there is no direct linear ranking. For example, a partial hand consisting of four spades could turn into a flush if another spade is added (good hand) or nothing in particular if a non-spade is added (bad hand).

One way to score partial hands is to use the average rank of all possible complete hands that the partial hand can develop into. Partial hands can then be binned according to their assigned scores. Note that this is only an approximation and a convenient way of grouping hands into bins; we are not actually using these scores to judge hands. We play each bin against another to get the expected payoff, and this is used in defining the payoffs for the "abstracted" game.

We introduced another game to test our method for dealing with games with potential. For this game, we use a 52-card deck. Each player is dealt one private card. A third "public" card is dealt onto the table to be shared by both players.

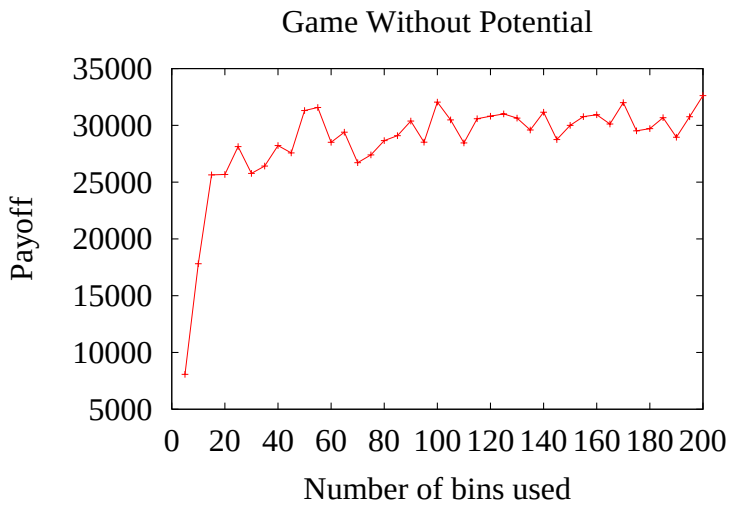

Fig. 2. Payoff vs. optimal for a game with betting only at the end

Players then bet (up to three raises). A final public card is then dealt and the player with the best 3-card hand (one private and two public cards) wins the pot.[2] For this game, $(b, c, d, r) = (1, 4, 52, 3)$, so the game tree has over 46M leaves.

Although this game tree would be too large to solve with our available software, we can eliminate two of the cards from the computation. First, because one public card is dealt before any betting takes place, it is *public knowledge* and we can essentially treat this card as part of the problem statement (assuming we are willing to solve a linear program at game time). Second, because the other public card is dealt after betting is complete, we can compute the *expected value* of the winnings over all possible cards when we compute the values for the leaves. Therefore, we really only have to reason specifically about the two hidden cards, or a game tree with around 21K leaves. Figure 3 shows the result of player 2 versus player 1 (player 2 is using the optimal strategy). Similar to the game without potential, we can do quite well using a very small number of bins.

In a game with multiple betting rounds, a different binning scheme would be used at each round, with transition probabilities calculated between consecutive pairs of bins. These probabilities can be calculated by enumeration (costly) or by random sampling (risky). For a game with b betting rounds at r raises per

[2] Three-card hands are ranked slightly differently from five-card hands: three of a kind beats a straight, and a straight beats a flush. See `http://conjelco.com/faq/poker.html#P15` for details.

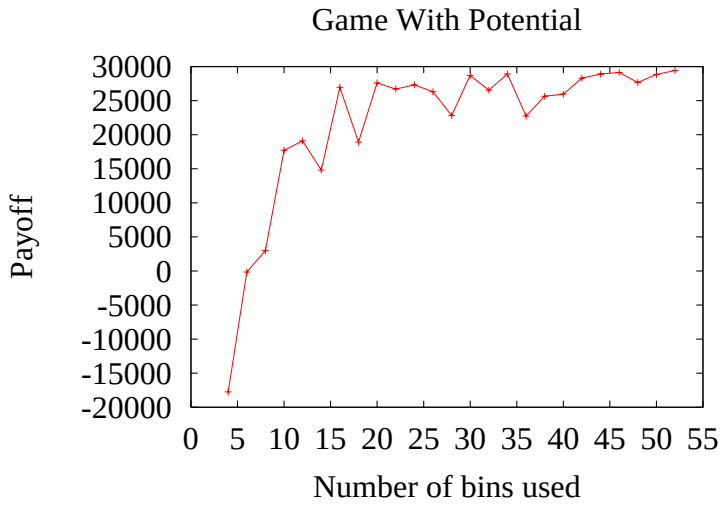

Fig. 3. Payoff vs. optimal for a game with hand potential

round with B bins used between each round, the abstracted game tree has $B^b 2^{rb}$ leaves. This can be a substantial reduction if $b << c$ (many cards dealt between each betting round) or $B << d$ (far fewer bins than unique cards in the deck).

4 Independent Betting Rounds

Texas Hold'em is a poker variant used to decide the world champion. Each player is dealt two cards, followed by a betting round, then three public cards are dealt, followed by a betting round, then one public card, followed by a betting round, then one final public card and betting round. Thus, the game-tree size for Texas Hold'em is huge.

In the context of our current work, we introduce *Rhode Island Hold'em*, which is intended to be similar to Texas Hold'em in style, but much smaller. In Rhode Island Hold'em, each player is dealt one private card, followed by a betting round, followed by a public card, followed by a betting round, followed by a final public card and betting round. The three-card hands based on the private card and two public cards are compared. Although this is clearly a much smaller game than Texas Hold'em, at $(b, c, d, r) = (3, 4, 52, 3)$ the game tree still has more than 2.9B leaves.

Although the binning idea from the previous section could be applied to these games with perhaps between 5 and 10 bins, we decided to explore a different type of approximation. Whereas binning techniques are useful in controlling the combinatorial explosion due to the number of different hands that can be dealt,

the techniques in this section address the explosion due to the large number of betting rounds. The main idea is to treat betting rounds quasi-independently instead of solving the entire game at once.

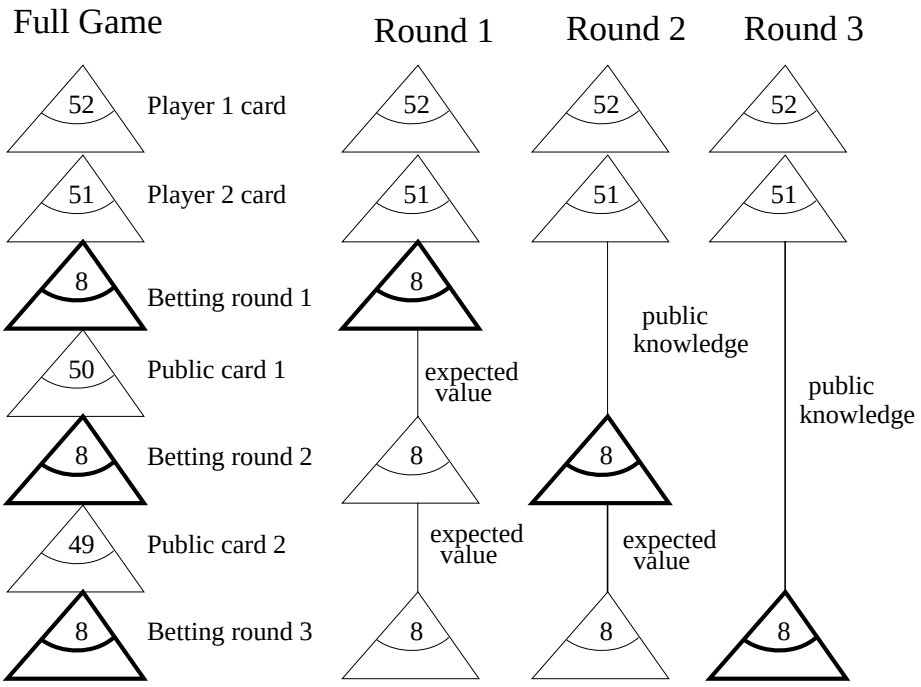

Fig. 4. Approximating Rhode Island Hold'em by handling betting rounds independently (using expected value for future cards, filling in public knowledge from previous rounds)

Figure 4 illustrates the process of handling the betting rounds independently. We replicate the full game tree three times, once for each betting round. To solve the first betting round, we make the assumption that our decisions in the first betting round are not very sensitive to whether future betting decisions come before or after additional cards are dealt. By moving the dealing of the public cards to the end of the game, we can then compute expected values over the dealt cards and remove the corresponding branches from the game tree.

The average expected payoff can be calculated by exhaustively going through each node. This works for a game such as Rhode Island Hold'em. But, for a larger game such as Texas Hold'em, a randomized simulation can be performed to estimate the average expected payoff.

Using this technique, the game tree for solving the first betting round now contains around 1.4M leaves and can be solved offline in approximately 15 min-

utes. We are currently uncertain regarding the accuracy of the strategy obtained using this technique. Because no information about past betting decisions is lost, we hypothesize that the strategy found this way is optimal or very close to it.

To solve the game tree for the second betting round, we need to build the part of the tree that represents the previous round as well. Again, we need to deal with the branching caused by future chance events (outcome uncertainty). We have chosen the alternative of completely ignoring the previous round, so we can keep our game tree at a reasonable size. The tradeoff here is the loss of information (the decisions made by the players are lost and those may reveal information about the opponent's hidden cards). However, the resulting game tree only has around 170K leaves and can be solved in a few seconds. The third round is even smaller (around 21K leaves) and is solved in under a second.

5 Implementation and Evaluation

Using the techniques from the previous section, we have reduced the game tree(s) for Rhode Island Hold'em to a manageable size. However, there is a danger that the approximation produced will lead to strategies far from the optimal play. To evaluate the performance of our program, we designed several opponents for it to compete against. First, we designed a rule-based player (RB), detailed in the Appendix,, that emulates how a good human player might play the game of Rhode Island Hold'em. The program generates randomized strategies depending on the state of the game. Next, we modified the rule-based program to model different player behaviors. Some players are willing to risk more (risk-seeking players) and pursue more aggressive strategies (AG), while others are more conservative (CON) and do not risk much (risk-averse players). We use appropriate rules to model both types.

We also have an opponent-modeling program (OPP) to play the game of Rhode Island Hold'em. This program plays by assuming its opponent is RB. It uses its knowledge of RB's behavior to compute a probability distribution over the hidden card given the observed betting behavior. It then makes choices to maximize its expected winnings (calculated by random sampling). This is a strategy used in the successful Loki program for playing Texas Hold'em [1].

Using the five opponents, we evaluated the effectiveness of our program. A thousand games were played between our program and each of the five opponents. During the competition, each program played 500 games as player 1 and the other 500 as player 2. This is done so that no one program has the advantage due to position. We felt 1,000 total games, which corresponds to roughly 30 hours of poker playing, would make for a meaningful comparison. The results, in terms of units gained, are summarized in Table 1.

From Table 1, we see that our program beat all of the opponents. The opponent modeling program was the toughest competitor, but the system was still able to win on average, to the tune of 5.4 units per 100 games. It is interesting to see that the aggressive player (AG) and the conservative player (CON) both did worse than the normal rule-based system (RB). AG lost more money proba-

Table 1. Units gained per 100 hands by our system versus opponents

	RB	AG	CON	OPP
Result	+8.3	+12.7	+9.1	+5.4

RB = Rule-based (Normal) AG = Rule-based (Aggressive)
CON = Rule-based (Conservative) OPP= Opponent-modeling

bly due to its aggressiveness in betting and bluffing in inappropriate situations. CON lost more because it did not take advantage of bluffing situations and also probably got bluffed out of many hands.

We analyzed the program's play and found that it was successfully using standard strategies such as bluffing and slow playing. One unexpected aspect was early round bluffing: human players are usually advised against bluffing in the early rounds of play with a low potential hand, whereas the program was bluffing in these situations. Because of the approximation due to computing independent betting rounds, it was possible that this behavior was a suboptimal artifact. To try to rule this out, we generated a new strategy that matched the computed strategy everywhere, but inhibited first and second round bluffing with low potential hands.[3] As player 2, this strategy scored +8.5 against RB, whereas the computed strategy scored +11.3, suggesting that this type of bluffing is a valuable strategy in this game.

6 Conclusion

Our program did an excellent job of playing the game of Rhode Island Hold'em. We would like to see it scale up to play a more complex game such as Texas Hold'em. Some of the main challenges that remain to be solved are the coarseness of the derived strategy from using a limited number of bins in the computation (probably between 5 and 10) and the fact that the system needs to either cache a large number of precomputed games or solve large linear programs during a match. Also, most Texas Hold'em games are played between 7 to 12 players, and currently, we are not aware of game-theoretic approaches for solving large multi-player games of imperfect information.

Games are excellent places to explore new ideas because of their clearly defined rules, specific goals, and the opportunity they present for comparison to human experts. In the past, game-playing research focused on deterministic games of perfect information. But, in computer science and emerging electronic commerce applications, many problems are made difficult by unreliable and imperfect information. It is our hope that by tackling a stochastic game with imperfect

[3] We considered replacing these bluffs with an equal number of semi-bluffs from stronger hands, but since the program is already semi-bluffing, we thought additiona semi-bluffs would only hurt its performance further.

information such as poker, we can learn more about dealing with uncertainty and apply our abstraction ideas to other similar problems.

A Rule-Based System for Rhode Island Hold'em

For Player 1 (who goes first), B represents the probability of betting, PB represents the probability of betting after Player 1 passes and Player 2 bets. For Player 2, pB represents the probability of betting after seeing a pass and bB represents the probability of betting after seeing a bet.

Round 1

Player 1 Holding	*Player 1 B*	*Player 1 PB*
2,3,4,5	0.05	0
6,7,8,9	0.4	0.1
10,J	0.9	0.9
Q,K,A	1	1
Player 2 Holding	*Player 2 pB*	*Player 2 bB*
2,3,4	0.3	0
5,6,7,8	0.7	0.4
9,10	0.9	0.8
J,Q,K,A	1	1

Round 2

Player 1 Holding	*Player 1 B*	*Player 1 PB*
Pair	1	1
2 card straight flush	0.9	1
2 card straight	0.7	0.7
2 card flush (9-A)	0.8	0.8
2 card flush (6-8)	0.6	0.6
2 card flush (2-5)	0.4	0.4
high card (J-A)	0.5	0.5
high card (7-10)	0.5	0.4
high card (5-6)	0.3	0.1
high card (2-4)	0.1	0

Player 2 Holding	*Player 2 pB*	*Player 2 bB*
Pair	1	1
2 card straight flush	1	1
2 card straight	0.9	0.7
2 card flush (9-A)	1	1
2 card flush (6-8)	0.8	0.6
2 card flush (2-5)	0.8	0.4
high card (J-A)	0.8	0.5
high card (7-10)	0.6	0.4
high card (5-6)	0.3	0.1
high card (2-4)	0.0	0.0
Round 3		
Player 1 Holding	*Player 1 B*	*Player 1 PB*
straight Flush	1	1
three of a kind	1	1
straight	1	1
flush (9-A)	1	1
flush (6-8)	0.8	0.8
flush (4-5)	0.7	0.8
flush (2-3)	0.4	0.4
pair	1	1
(we have the hole card that makes up the higher pair, no flush possible)		
pair	0.9	1
(we have the hole card that makes up the lower pair, no flush possible)		
pair	0.8	1
(we have the hole card that makes up the higher pair, flush possible)		
pair	0.5	0.9
(we have the hole card that makes up the lower pair, flush possible)		
pair (kicker J-A)	0.8	0.8
(pair on the board)		
pair (kicker 8-10)	0.8	0.6
pair (kicker 5-7)	0.4	0.2
pair (kicker 2-4)	0.2	0
high card (J-A)	0.8	0.8
(no flush possible)		
high card (8-10)	0.7	0.5
(no flush possible)		
high card (5-7)	0.3	0.2
(no flush possible)		
high card (2-4)	0.1	0
high card (A)	0.8	0.8
(flush possible)		
high card (J-K)	0.7	0.7
(flush possible)		
high card (8-10)	0.5	0.3
(flush possible)		
high card (2-7)	0.2	0
(flush possible)		

Player 2 Holding	*Player 2 pB*	*Player 2 bB*
straight Flush	1	1
three of a kind	1	1
straight	1	1
flush (9-A)	1	1
flush (6-8)	1	0.8
flush (4-5)	0.8	0.5
flush (2-3)	0.5	0.3
pair	1	1
(we have the hole card that makes up the higher pair, no flush possible)		
pair	1	1
(we have the hole card that makes up the lower pair, no flush possible)		
pair	1	1
(we have the hole card that makes up the higher pair, flush possible)		
pair	1	0.8
(we have the hole card that makes up the lower pair, flush possible)		
pair (kicker J-A)	1	0.8
(pair on the board)		
pair (kicker 8-10)	0.9	0.5
pair (kicker 5-7)	0.2	0.1
pair (kicker 2-4)	0	0
high card (J-A)	0.9	0.8
(no flush possible)		
high card (8-10)	0.7	0.3
(no flush possible)		
high card (5-7)	0.3	0
(no flush possible)		
high card (2-4)	0	0
high card (A)	0.9	0.8
(flush possible)		
high card (J-K)	0.8	0.7
(flush possible)		
high card (8-10)	0.5	0.3
(flush possible)		
high card (2-7)	0.2	0
(flush possible)		

References

1. Darse Billings, Denis Papp, Jonathan Schaeffer, and Duane Szafron. Opponent modeling in poker. In *Proceedings of the 15th National Conference on Artificial Intelligence*, pages 493–499, 1998.
2. Leslie Pack Kaelbling, Michael L. Littman, and Andrew W. Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996.
3. Daphne Koller, Nimrod Megiddo, and Bernhard von Stengel. Efficient computation of equilibria for extensive two-person games. *Games and Economic Behavior*, 14(2):247–259, 1996.

4. Daphne Koller and Avi Pfeffer. Representations and solutions for game-theoretic problems. *Artificial Intelligence*, 94(1–2):167–215, 1997.
5. Avi Pfeffer, Daphne Koller, and Ken T. Takusagawa. State-space approximations for extensive form games. Workshop paper at First World Congress on Game Theory, 2000.
6. Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, 1998.
7. J. von Neumann and O. Morgenstern. *Theory of Games and Economic Behavior*. Princeton University Press, Princeton, NJ, 1947.