An Exploitative Monte-Carlo Poker Agent

Immanuel Schweizer, Kamill Panitzek, Sang-Hyeun Park, and Johannes Fürnkranz

TU Darmstadt, Knowledge Engineering Group,
D-64289 Darmstadt, Germany

Abstract. We describe the poker agent AKI-REALBOT which participated in the 6-player Limit Competition of the third Annual AAAI Computer Poker Challenge in 2008. It finished in second place, its performance being mostly due to its superior ability to exploit weaker bots. This paper describes the architecture of the program and the Monte-Carlo decision tree-based decision engine that was used to make the bot's decision. It will focus the attention on the modifications which made the bot successful in exploiting weaker bots.

1 Introduction

Poker is a challenging game for AI research because of a variety of reasons [3]. A poker agent has to be able to deal with *imperfect* (it does not see all cards) and *uncertain* information (the immediate success of its decisions depends on random card deals), and has to operate in a *multi-agent* environment (the number of players may vary). Moreover, it is not sufficient to be able to play an optimal strategy (in the game-theoretic sense), but a successful poker agent has to be able to exploit the weaknesses of the opponents. Even if a game-theoretical optimal solution to a game is known, a system that has the capability to model its opponent's behavior may obtain a higher reward. Consider, for example, the simple game of *rock-paper-scissors* aka *RoShamBo* [1], where the optimal strategy is to randomly select one of the three possible moves. If both players follow this strategy, neither player can gain by unilaterally deviating from it (i.e., the strategy is a *Nash equilibrium*). However, against a player that always plays *rock*, a player that is able to adapt its strategy to always playing *paper* can maximize his reward, while a player that sticks with the "optimal" random strategy will still only win one third of the games. Similarly, a good poker player has to be able to recognize weaknesses of the opponents and be able to exploit them by adapting its own play. This is also known as *opponent modeling*.

In every game, also called a *hand*, of fixed limit Texas Hold'em Poker, there exist four game states. At the *pre-flop* state, every player receives two *hole cards*, which are hidden to the other players. At the *flop*, *turn* and *river* states, three, one and one *community cards* are dealt face up respectively, which are shared by all players. Each state ends with a *betting* round. At the end of a hand (the *showdown*) the winner is determined by forming the strongest possible five-card poker hand from the players's hole cards and the community cards. Each game begins by putting two forced bets (*small blind* and *big blind*) into the pot, where the big blind is the minimal betting amount, in this context also called *small bet* (SB). In pre-flop and flop the betting amount is restricted to SBs, whereas on turn and river one has to place a *big bet* (2× SB). At every turn, a player can either *fold*, *check/call* or *bet/raise*.

B. Mertsching, M. Hund, and Z. Aziz (Eds.): KI 2009, LNAI 5803, pp. 65–72, 2009.
© Springer-Verlag Berlin Heidelberg 2009

In this paper, we will succinctly describe the architecture of the AKI-REALBOT poker playing engine (for more details, cf. [7]), which finished second in the AAAI-08 Computer Poker Challenge in the 6-player limit variant. Even though it lost against the third and fourth-ranked player, it made this up by winning more from the fifth and sixth ranked player than any other player in the competition.

2 Decision Engine

2.1 Monte-Carlo Search

The Monte Carlo method [6] is a commonly used approach in different scientific fields. It was successfully used to build AI agents for the games of bridge [5], backgammon [8] and Go [4]. In the context of game playing, its key idea is that instead of trying to completely search a given game tree, which is typically infeasible, one draws random samples at all possible choice nodes. This is fast and can be repeated sufficiently frequently so that the average over these random samples converges to a good evaluation of the starting game state.

Monte-Carlo search may be viewed as an orthogonal approach to the use of evaluation functions. In the latter case, the intractability of exhaustive search is dealt with by limiting the search depth and the use of an evaluation function at the leaf nodes, whereas Monte Carlo search deals with this problem by limiting the search breadth at each node and the use of random choice functions at the decision nodes. A key advantage of Monte Carlo search is that it can deal with many aspects of the game without the need for explicitly representing the knowledge. Especially for poker, these include *hand strength*, *hand potential*, *betting strategy*, *bluffing*, *unpredictability* and *opponent modeling* [2]. These concepts, for which most are hard to model explicitly, are considered implicitly by the outcome of the simulation process.

In each game state, there are typically three possible actions, *fold*, *call* and *raise*.[1] AKI-REALBOT uses the simulated expected values (EV) for them to evaluate a decision. These EVs are estimated by applying two independent Monte-Carlo searches, one for the call action and the other one for the raise action (cf. Figure 1). Folding does not have to be simulated, since the outcome can be calculated immediately. Then at some point, these search processes are stopped by the Time Management component [7], which tries to utilize the available time as effective as possible. Since an increase in the number of simulated games also increases the quality of the EVs and therefore improves the quality of the decision, a multi-threading approach was implemented.

Our Monte-Carlo search is not based on a uniformly distributed random space but the probability distribution is biased by the previous actions of a player. For this purpose, AKI-REALBOT collects statistics about each opponent's probabilities for folding (f), calling (c), and raising (r), thus building up a crude opponent model. This approach was first described in [2] as selective sampling. For each played hand, every active opponent player is assigned hole cards. The selection of the hole cards is influenced by the opponent model because the actions a player takes reveal information about the strength of his cards, and should influence the sample of his hole cards. This selection

[1] The AAAI rules restrict the number of bets to four, so that a *raise* is not always possible.

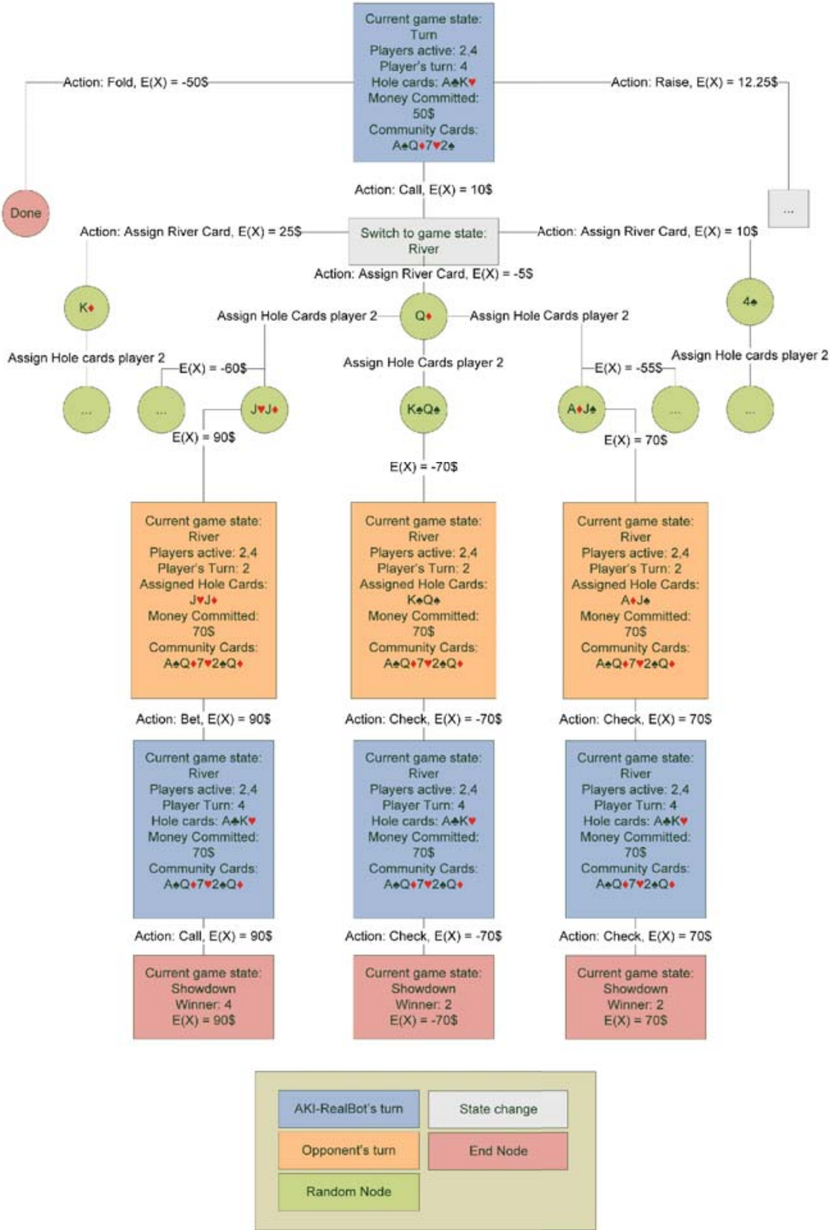


Fig. 1. Monte Carlo Simulation: the figure depicts an example situation on the turn, where AKI-REALBOT is next to act (top). The edges represent in general the actions of players or that of the chance player. For the decisions *call* or *raise* (middle and right path), two parallel simulations are initiated. The path for the *call* decision for example (in the middle), simulates random games until the showdown (the river card is Qd, the opponent cards are estimated as KsQs, and both players check on the river.) and the estimated loss of 70$ is backpropagated along the path.

is described in detail in Section 3. After selecting the hole cards, at each player's turn, a decision is selected for this player, according to a probability vector (f, c, r), which are estimated from the previously collected data.

Each community card that still has to be unveiled is also randomly picked whenever the corresponding game state change happens. Essentially the game is played to the showdown. The end node is then evaluated with the amount won or lost by AKI-REALBOT, and this value is propagated back up through the tree. At every edge the average of all subtrees is calculated and represents the EV of that subtree. Thus, when the simulation process has terminated, the three decision edges coming from the root node hold the EV of that decision. In a random simulation, the better our hand is, the higher the EV will be. This is still true even if we select appropriate samples for the opponents' hole cards and decisions as long as the community cards are drawn uniformly distributed.

2.2 Decision Post-processing

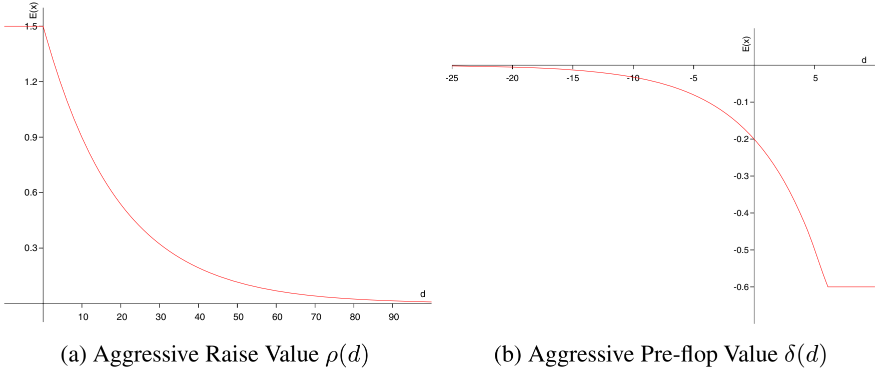
AKI-REALBOT post-processes the decision computed by the Monte-Carlo search in order to increase the adaptation to different agents in a multiplayer scenario even further with the goal of exploiting every agent as much as possible (in contrast to [2]). The exploitation of weak opponents is based on two simple considerations:

1. Weak players play too tight, i.e. they fold too often
2. Weak players play too loose (especially post-flop), which is the other extreme: they play too many marginal hands until the showdown

These simply defined weak players can be easily exploited by an overall aggressive play strategy. It is beneficial for both types of players. First, if they fold too often, one can often bring the opponent to fold a better hand. Second, against loose players, the hand strength of marginal hands increase, such that one can win bigger pots with them than usual. Besides the aggressive play, the considerations imply a loose strategy. By expecting that AKI-REALBOT can *outplay* the opponent, it tries to play as many hands as possible against weaker opponents.

This kind of commonly known expert-knowledge was explicitly integrated. For this purpose, so-called *decision bounds* were imposed on the EVs given by the simulation. This means that for every opponent, AKI-REALBOT calculates dynamic upper and lower bounds for the EV, which were used to alter the strategy to a more aggressive one against weaker opponents. $E(f)$ will now denote the EV for the *fold* path, while $E(c)$ and $E(r)$ will be the values for *call* and *raise* respectively. Without post-processing, AKI-REALBOT would pick the decision x where $E(x) = \max_{i=\{f,c,r\}} E(i)$.

Aggressive Pre-flop Value. The lower bound is used for the pre-flop game state only. As long as the EV for folding is smaller than the EV for either calling or raising (i.e., $E(f) < \max(E(c), E(r))$), it makes sense to stay in the game. More aggressive players may even stay in the game if $E(f) - \delta < \max(E(c), E(r))$ for some value $\delta > 0$. If AKI-REALBOT is facing a weak agent W it wants to exploit its weakness. This means that AKI-REALBOT wants to play more hands against W. This can be achieved by setting $\delta > 0$. We assume that an agent W is weak if he has lost money against AKI-REALBOT over a fixed period of rounds. For this purpose, AKI-REALBOT maintains



(a) Aggressive Raise Value $\rho(d)$ (b) Aggressive Pre-flop Value $\delta(d)$

Fig. 2. Dynamic Decision Bounds

a statistic over the number of small bets (SB) d, that has been lost or won against W in the last $N = 500$ rounds. For example, if W on average loses 0.5 SB/hand to AKI-REALBOT then $d = 0.5 \times 500 = 250$ SB. Typically, d is in the range of $[-100, 100]$. Then, the *aggressive pre-flop value* δ for every opponent is calculated as

$$\delta(d) = \max(-0.6, -0.2 \times (1.2)^d)$$

Note that $\delta(0) = -0.2$ (SB), and that the value of maximal aggressiveness is already reached with $d \approx 6$ (SB). That means, that AKI-REALBOT already sacrifices in the initial status $d = 0$ some EV (maximal -0.2 SB) in the pre-flop state, in the hope to outweight this drawback by *outplaying* the opponent post-flop. Furthermore, if AKI-REALBOT has won in the last 500 hands only more than 6 SB against the faced opponent, it reaches its maximal *optimism* by playing also hands which EVs were simulated as low as ≈ -0.6 SB. This makes AKI-REALBOT a very aggressive player pre-flop, especially if we consider that δ for more than one active opponent is calculated as the average of their respective δ values.

Aggressive Raise Value. The upper bound is used in all game states and makes AKI-REALBOT aggressive on the other end of the scale. As soon as this upper bound is reached, it will force AKI-REALBOT to raise even if $E(c) > E(r)$. This will increase the amount of money that can be won if AKI-REALBOT is very confident about his hand strength. This upper bound is called the *aggressive raise value* ρ.

$$\rho(d) = \min(1.5, 1.5 \times (0.95)^d)$$

Here, the upper bound returns $\rho(0) = 1.5$ for the initial status $d = 0$, which is 1.5 times the SB and therefore a very confident EV. In fact, it is so confident that this is also the maximum value for ρ. The aggressive raise value is not influenced if we lose money against a player. If, on the other hand, AKI-REALBOT wins money against an agent W, it will slowly converge against zero, resulting in a more and more aggressive play.

As said before, the value of d is calculated based on a fixed amount of past rounds. It is therefore continuously changing with AKI-REALBOT's performance over the past rounds. The idea is to adapt dynamically to find an optimal strategy against any

single player. On the other hand, it is easy to see that this makes AKI-REALBOT highly vulnerable against solid, strong agents.

3 Opponent Modeling

In general, the opponent modeling of the AKI-REALBOT, which is used to adjust the implemented Monte-Carlo simulation to the individual behavior of the other players, considers every opponent as a straight-forward player. That means, we assume that aggressive actions indicate a high hand strength and passive actions a low hand strength. Within the simulation, the opponent's hand strength is guessed based on the action he takes. So if a player often folds in the pre-flop phase but calls or even raises in one special game this means he has probably a strong hand. In addition the opponent modeling tries to map cards to the actions every player takes.

AKI-REALBOT has two different routines that enables it to guess hole cards according to the opponent model. Which routine is used depends on the game state.

Pre-Flop State: In pre-flop, we assume that the actions of a player are only based on his hole cards. He is either confident enough to raise or to make a high call, whereas making a small call may indicate a lower confidence in his hand. A high call is indicated by committing more than a big bet. In either case his observed fold ratio f and call ratio c are used to calculate an upper and lower bound for the set of hole cards. It is common to divide the set of possible hole cards into *buckets*, where each bucket consists of hole cards of similar hand strength, to reduce the space of hole card combinations. We used five buckets, U_0 being the weakest bucket and U_4 (e.g. containing the cards AA) the strongest. The buckets have the following probability distribution: $p(U_0) = 0.65,\ p(U_1) = 0.14,\ p(U_2) = 0.11,\ p(U_3) = 0.07,\ p(U_4) = 0.03$.

In the first case of the above example, the upper bound U is set to the maximum possible bucket value ($U_h = 4$) because high confidence was shown. The lower bound is calculated by taking $l = c + f$ and relating this to the bucket. That means, the lower bound is set to exclude the hole cards, for which the player would only call or fold. If for example $f = 0.71$ and $c = 0.2$, the player raises only in 9% of cases. Since we assumed a straight-forward or honest player, we imply that he only does this with the top 9% of hole cards. So, the lower bound is set to U_3. Then, the hole cards for that player are selected randomly from the set of hole cards which lie between the bounds.

Post-Flop State. The second routine for guessing the opponents' hole cards is used when the game has already entered a post-flop state. The main difference is that the actions a player takes are now based on both hidden (his hole cards) and visible information (the board cards). Therefore, AKI-REALBOT has to estimate the opponent's strength also by taking the board cards into account. It estimates how much the opponent is influenced by the board cards. This is done by considering the number of folds for the game state flop. If a player is highly influenced by the board he will fold often on the flop and only play if his hand strength has increased with the board cards or if his starting hand was irrespectively very strong.

This information is used by AKI-REALBOT to assign hole cards in the post-flop game state. Two different methods are used here:

– *assignTopPair*: increases the strength of the hole cards by assigning the highest rank possible, i.e., if there is an ace on the board the method will assign an ace and a random second card to the opponent.
– *assignNutCard*: increases the strength of the hole cards even more by assigning the card that gives the highest possible poker hand using all community cards i.e. if there is again an ace on the board but also two tens the method will assign a ten and a random second card.

These methods are used for altering one of the player's hole card on the basis of his fold ratio f on the flop. We distinguish among three cases based on f, where probability values p_{Top} and p_{Nut} are computed.

(1) $f < \frac{1}{3} \Rightarrow p_{Top} = 3(f)^2 \in [0, \frac{1}{3}[\quad p_{Nut} = 0$
(2) $\frac{1}{3} \leq f < \frac{2}{3} \Rightarrow p_{Top} = \frac{1}{3} \quad p_{Nut} = \frac{1}{3}(3f-1)^2 \in [0, \frac{1}{3}[$
(3) $f \geq \frac{2}{3} \Rightarrow p_{Top} = \frac{1}{3} \quad p_{Nut} = f - \frac{1}{3} \in [\frac{1}{3}, \frac{2}{3}]$

To be clear, *assignTopPair* is applied with a probability of p_{Top}, *assignNutCard* is applied with a probability of p_{Nut} and with a probability of $1 - (p_{Top} + p_{Nut})$ the hole cards are not altered. As one can see in the formulas, the higher f is, the more likely it is that the opponent will be assigned a strong hand in relation to the board cards. Note that for both methods the second card is always assigned randomly. This will sometimes strongly underestimate the cards e.g. when there are three spade cards on the board *assignNutCard* will not assign two spade cards.

4 AAAI-08 Computer Poker Competition Results

AKI-REALBOT participated in the 6-player Limit competition part of the Computer Poker Challenge at the AAAI-08 conference in Chicago. There were six entries: HYPERBOREAN08_RING aka POKI0 (University of Alberta), DCU (Dublin City University), CMURING (Carnegie Mellon University), GUS6 (Georgia State University), MCBOTULTRA and AKI-REALBOT, two independent entries from TU Darmstadt.

Among these players, 84 matches were played with different seating permutations so that every bot could play in different positions. Since the number of participants were exactly 6, every bot was involved in all 84 matches. In turn, this yielded 504000 hands for every bot. In that way, a significant result set was created, where the final ranking was determined by the accumulated win/loss of each bot over all matches.[2]

Table 1 shows the results over all 84 matches. All bots are compared with each other and the win/loss statistics are shown in SBs. Here it becomes clear that AKI-REALBOT exploits weaker bots because the weakest bot, GUS6, loses most of it's money to AKI-REALBOT. Note, that GUS6 lost in average more than 1.5 SBs per hand, which is a worse outcome than by folding every hand, which results in an avg. loss of 0.25 SB/Hand. Although AKI-REALBOT loses money to DCU and CMURING it manages to rank second, closely behind POKI0, because it is able to gain much higher winnings against the weaker players than any other player in this field. Thus, if GUS6 had not participated in this competition, AKI-REALBOT's result would have been much worse.

[2] The official results can be found at
http://www.cs.ualberta.ca/~pokert/2008/results/

Table 1. AAAI-08 Poker Competition Results: pairwise and overall performance of each entry

	POKI0	AKI-REAL	DCU	CMURING	MCBOT	GUS6
POKI0		65,176	2,655	18,687	29,267	214,840
AKI-REALBOT	-65,176		-15,068	-2,769	30,243	348,925
DCU	-2,665	15,068		7,250	16,465	90,485
CMURING	-18,687	2,769	-7,250		7,549	92,453
MCBOTULTRA	-29,267	-30,243	-16,465	-7,549		16,067
GUS6	-214,840	-348,925	-90,485	-92,453	-16,067	
Total	330,822	296,293	126,657	76,848	-67,529	-763,091
avg. winnings/game	3934	3579	1512	939	-800	-9042
SB/Hand	0.656	0.588	0.251	0.152	-0.134	-1.514
Place	1.	2.	3.	4.	5.	6.

5 Conclusion

We have described the poker agent AKI-REALBOT that finished second in the AAAI-08 Poker Competition. Its overall performance was very close to the winning entry, even though it has lost against three of its opponents in a direct comparison. The reason for its strong performance was its ability to exploit weaker opponents. In particular against the weakest entry, it won a much higher amount than any other player participating in the tournament. The key factor for this success was its very aggressive opponent modeling approach, due to the novel adaptive post-processing step, which allowed it to stay longer in the game against weaker opponents as recommended by the simulation.

Based on these results, one of the main further steps is to improve the performance of AKI-REALBOT against stronger bots. An easy way would be to adopt the approaches of the strongest competitors of the competition, for which there exists a multitude of publications. But, we see also yet many possible improvements for our exploitative approach, which we elaborate in [7] and are currently working on.

References

1. Billings, D.: Thoughts on RoShamBo. ICGA Journal 23, 3–8 (2000)
2. Billings, D., Castillo, L.P., Schaeffer, J., Szafron, D.: Using probabilistic knowledge and simulation to play poker. In: AAAI/IAAI, pp. 697–703 (1999)
3. Billings, D., Davidson, A., Schaeffer, J., Szafron, D.: The challenge of poker. Artif. Intell. 134(1-2), 201–240 (2002)
4. Bouzy, B.: Associating domain-dependent knowledge and Monte Carlo approaches within a Go program. In: Joint Conference on Information Sciences, pp. 505–508 (2003)
5. Ginsberg, M.L.: Gib: Steps toward an expert-level Bridge-playing program. In: Proc. of the 16th International Joint Conference on Artificial Intelligence (IJCAI 1999), pp. 584–589 (1999)
6. Metropolis, N., Ulam, S.: The Monte Carlo method. J. Amer. Stat. Assoc. 44, 335–341 (1949)
7. Schweizer, I., Panitzek, K., Park, S.-H., Fürnkranz, J.: An exploitative Monte-Carlo poker agent. Technical Report TUD-KE-2009-02, TU Darmstadt, Knowledge Engineering Group (2009), `http://www.ke.informatik.tu-darmstadt.de/publications/reports/tud-ke-2009-02.pdf`
8. Tesauro, G.: Temporal difference learning and TD-Gammon. Commun. ACM 38(3), 58–68 (1995)