ECAI 2008
M. Ghallab et al. (Eds.)
IOS Press, 2008

doi:10.3233/978-1-58603-891-5-458

Adaptive play in Texas Hold'em Poker

Raphaël Maîtrepierre and **Jérémie Mary** and **Rémi Munos**[1]

Abstract. We present a Texas Hold'em poker player for limit heads-up games. Our bot is designed to adapt automatically to the strategy of the opponent and is not based on Nash equilibrium computation. The main idea is to design a bot that builds beliefs on his opponent's hand. A forest of game trees is generated according to those beliefs and the solutions of the trees are combined to make the best decision. The beliefs are updated during the game according to several methods, each of which corresponding to a basic strategy. We then use an exploration-exploitation bandit algorithm, namely the UCB (Upper Confidence Bound), to select a strategy to follow. This results in a global play that takes into account the opponent's strategy, and which turns out to be rather unpredictable. Indeed, if a given strategy is exploited by an opponent, the UCB algorithm will detect it using change point detection, and will choose another one.

The initial resulting program , called Brennus, participated to the AAAI'07 Computer Poker Competition in both online and equilibrium competition and ranked eight out of seventeen competitors.

1 INTRODUCTION

Games are an interesting domain for Artificial Intelligence research. Computer programs are better than humans in many games including Othello, chess [10] or checkers [14]. Those games are perfect information games in the sense that all useful informations for predicting the outcome of the game is common knowledge of all players. Moreover, those games are deterministic. On the contrary, Poker is an incomplete information and stochastic game: players do not know which cards their opponents are holding neither the community cards remaining to come. These aspects make Poker a challenging domain for AI research [7].

Thanks to Nash's work [13] we know the existence of an equilibrium strategy (Nash equilibrium) for the 2 players game Poker. Now, since this is a zero-sum game, if a player plays according to this strategy, in average he will not lose. But a good Poker player should also be able to adapt to his opponent game in order to exploit possible weaknesses since the goal of Poker is to win the maximum of chips. Studies have been done in the domain of opponent exploitation for the game of RoShamBo [3, 4]. For this game, a Nash equilibrium consists in playing uniformly randomly all three actions (Rock, Paper, Scissors); this strategy does not lose (in average) since it is unpredictable, but it does not win either!

Indeed, against a player who always plays "Paper" for example, the expected payoff of the Nash strategy is null (1/3 lose, 1/3 win, 1/3 draw), whereas a player who could exploit his opponent would quickly find out that playing "Scissors" all the time is the best decision.

[1] INRIA LILLE NORD EUROPE, France, email: {raphael.maitrepierre,jeremie.mary,remi.munos}@inria.fr

In the game of poker, this idea remains true up to some extent: if an opponent bluffs too many times we should call him more often, and on the other hand, if he does not bluff often, we should be very careful. So it appears necessary to model our opponent strategy if we want to exploit his weaknesses and maximize our income.

In the last few years, Poker research have received a large amount of interest. One of the first approaches to build a Poker playing bot was based on simulations [8]. The logical next step was to compute Nash equilibria for Poker. In a zero-sum game, it is possible to compute an equilibrium of the sequence form of the game using linear programming methods. But in Poker the state space is so huge that one needs to use abstraction methods (which gather similar states) before solving the sequence form game. Such powerful methods have been used in [5, 11, 12, 16] to reduce the size of the game and compute a near optimal Nash equilibrium. Some research has also been conducted in opponent modeling for game-tree search in [6], and the resulting program, named *Vexbot*, is also available in *Poker-Academy* software.

In this paper, we present a new method for building an adaptive Poker bot based on beliefs updates and strategies selection. Our contribution is two-fold. First, in opponent modeling we consider a belief update on the opponent's hand based on Bayes' rule, which combined with different opponent models, yields different basic strategies. Second, we consider a strategy selection procedure based on a bandit algorithm (namely the *Upper Confidence Bounds* algorithm introduced in [2]) which performs a good trading-off between **exploitation** (choose a strategy that has performed well against the opponent) and **exploration** (try another apparently sub-optimal strategy in order to learn more about the opponent).

The paper is organized as follows: after briefly reminding the rules of Hold'em Poker, we present our contributions in Section 3, with the description of the forest of game trees, the belief update rule for the opponent modeling, and the bandit algorithm for the strategy selection. We conclude with experimental results.

2 Rules of the game

In this paper we consider the two player version of Texas Hold'em Poker called *heads-up*. A good introduction to the rules can be found in [7]. The betting structure used is *limit poker*. This is the structure used in the AAAI Computer Poker Competition.

A hand of Hold'em consists in four stages, each one followed by a betting round.

The game begins with two forced bets called the blinds. The first player puts the small blind (half a small bet) and the other player puts the big blind (one small bet). Each player is dealt two hidden cards, a first betting round occurs, this is the preflop stage. Then, three community cards (called the *board*) are dealt face up, this is the flop stage. In the third stage, the turn, a fourth card is added to

the board. A last card is added to the board in the river stage. After the last betting round the showdown occurs: the remaining players compare their hands (their hole cards) and the player with the best five cards combination formed with his two cards and the community cards wins the *pot* (amount of chips bet by all players).

In limit Poker, two sizes of bet are used: in the first two stages the bet is called the small bet. And in the last ones the bet is called big bet and worths two small bets.

In betting rounds the player who acts has three possibilities:

- He may *fold*, so he loses the game and the chips he put in.
- He may *call*, in this case he puts the same amount of chips as his opponent, if no chips have been bet in the round this action is called *check*
- He may *raise*, in this case he puts one bet more than his opponent has bet, if no chips have been bet this action is called *bet*.

3 OUR APPROACH

The approach studied in this paper is close to the human way of playing poker: our bot tries to guess what are his opponent's hands based on the previous decisions of the opponent in this game. For that purpose, we assign to each possible hand of the opponent, the probability he holds this hand given what he has played before. This association hand/probability represents the beliefs of our bot, and are saved in a table. These probabilities are updated after each action taken by the opponent using a simple Bayes rule (see subsection 3.2). Then, given those beliefs, we compute a "forest" of Min-Max trees (where each tree corresponds to a possible hand assignments to both players based on the current beliefs of our bot about his opponent) to evaluate the current situation and make our decision based on a weighted combination of the solutions returned by the Min-Max trees. We described this step in the next subsection. This method is used to make decisions after the flop, for preflop play we use precalculated tables from [15].

3.1 Forest of Game Trees

A forest of trees is composed of a set of Min-Max game trees where each game tree is defined by two couples of hands, one for each player. A couple of hands represent a player point of view: his real hand and his belief about his opponent's hidden cards. For the AI player, real hand is represented by the two actual hidden cards dealt to him, and the opponent's hands are chosen randomly according to the current belief table of probabilities about his opponent's cards. For the opponent player, his real hand is chosen (randomly) according to the belief table (independently of the choice of the AI opponent's hand) and his belief about our bot's hand is uniformly randomly generated (i.e. currently, there is no model of the opponent's belief about our bot's cards). The beliefs about opponent's hands are fixed within a tree.

To each leaf and node of each game tree, 2 values are assigned, each one corresponding to the value for each player (V_{p1} and V_{p2}). One represents the expected outcome from the point of view of the AI bot: the result of the game between his hand against the current belief about his opponent (V_{p1}). The other value is the expected outcome from the point of view of the opponent (V_{p2}).

Since each possible hand for our opponent have different probabilities, we build a "forest" of such trees in order to evaluate the current situation. Once all the game trees have been solved, the value of each possible action is given by the convex combination of the values of all trees weighted by the probability of the hands used in the trees (belief that this tree corresponds to the true situation). Those probabilities are given by the beliefs table.

Each game tree is solved as follows. There are 3 kinds of nodes:

- Action nodes: Nodes representing actions of players.
- Chance nodes: Nodes representing chance events (cards dealing)
- Leaves: Nodes representing the end of the game.

The value of a leaf depends on the last selected action:

- If it is a "fold", the value corresponding to the player who has made this action is 0 while his opponent's value equals the amount of chips in the pot at this point of the tree,
- If it is a "call" the value of each player is the amount of chips won (or lost) against his opponent's hand.

Now, concerning the action nodes values computation: the value of the active player (the one who takes an action in that node) is defined as the maximal value (for the same player) of the 3 children nodes (corresponding to the 3 possible actions) minus the cost of the action (the amount of chips added in the pot corresponding to the action). His opponent's value is the value of the child corresponding to the action chosen by the active player. Figures 1 and 2 illustrate the action nodes value update. In case of equality, when choosing the max value for the active player, we choose the most aggressive action (i.e. "raise" rather than "call", "call" rather than "fold"), the reason being that in heads-up poker, playing aggressively is usually better than playing passively.

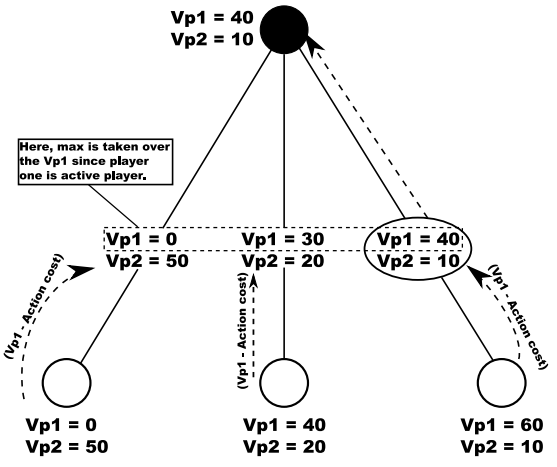

Figure 1. Update of action nodes: here active player is player 1 (black nodes). His values is V_{p1}. Values shown on the edges are the children nodes values minus the corresponding action cost. Active player choose the action corresponding to the maximum edge value. Here the rightmost action is chosen ($V_{p1} = 40$). His opponent value (V_{p2}) is the value corresponding to the action chosen by player1: $V_{p2} = 10$.

Players value at chance nodes is the mean of each sons of the node, for example if we are on the "turn" stage, in two players games, there is 45 remaining possible cards to be dealt. So value for players is:

$$V_{p1} = \frac{1}{45}\sum_{i=1}^{i=45} V_{p1_i}; V_{p2} = \frac{1}{45}\sum_{i=1}^{i=45} V_{p2_i}$$

Since computing whole trees is too long for online play, we use an approximation for computing trees values at chance nodes: instead

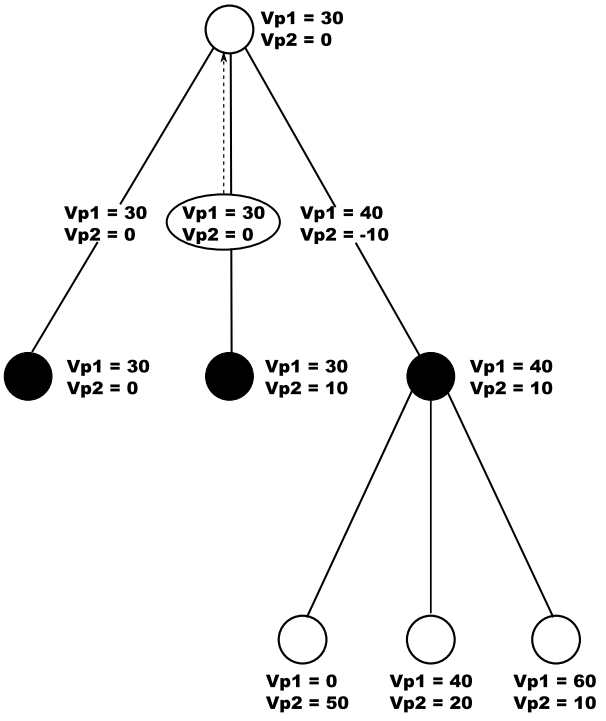

Figure 2. Here active player is player2 (white nodes), he chooses the max value among the V_{p2} (this corresponds to the second action). The corresponding V_{p1} and V_{p2} are updated.

of computing chance nodes values nine times at each stage of the game (1 time for each sequence of actions leading to a next stage) we compute values for the first chance node encountered, and for chance nodes in the same subtree (resulting from the same card dealt) we use the value of the first chance node, modified to consider the change of pot size.

3.2 Belief Update

The AI's belief about his opponent' hidden cards (H) is the probability that he really holds these cards given the past actions of the opponent. At the beginning of a hand[2], each possible couple of cards is assigned a uniform probability since no information is revealed from our opponent yet. After each action of our opponent, we update those beliefs according to a model of play of the opponent, which is expressed in terms of the probabilities of choosing an action given his game. Actually we consider several possible such models, each of which defining a specific style of play.

A model of play of our opponent is defined by the probabilities $\mathbb{P}(a|H, I_t)$ of choosing an action a given his hidden cards H and the information set I_t, where I_t represents all the information available to both players at time t (e.g. the flop, the bets of the players up to time t, ...). Now, once the opponent has chosen an action a at time t, the beliefs $\mathbb{P}(H|I_t)$ on his hidden cards H are updated according to Bayes' rule:

$$\mathbb{P}(H|I_t) = \mathbb{P}(H|I_{t-1})\mathbb{P}(a|H, I_{t-1}),$$

where $I_t = (I_{t-1} \cup \{a\})$.

[2] here hand means the game from preflop to showdown if it occurs

Thus a simple belief update is performed after each action, based on a model of play $\mathbb{P}(a|H, I_t)$ of the opponent. Now we explain our choice of such models. To define the style (or model) of play, Poker theorists usually [9] consider two attributes:

- Whether the player plays *tight* (plays very few hands) or *loose* (plays a lot of hands);
- Whether the player is *aggressive* (raises and bluffs) or *passive* (calls other players bets).

We selected three features to define relevant properties of a game state. The first one is the *stage S of the game* (Flop, Turn, River) since it defines the amount of the bets and the number of remaining community cards to come. The second one is the *hand strength F* (probability of winning) of the hand. The third one is the *size of the pot C* since it greatly influence the way of playing. We thus model the strategy of the opponent $\mathbb{P}(a|H, I_t)$ using these three features (F, C, S) of H and I_t, and write $\mathbb{P}(a|F, C, S)$ the corresponding model (approximation of $\mathbb{P}(a|H, I_t)$).

In our implementation, we model two basis strategies, one is tight/aggressive and the other is loose/passive. A model is defined as follow: for each possible stage of the game (Flop, Turn, River) we have a table that gives the probability of choosing each action as a function of the hand strength and the pot size. Hand strength is discretized into 5 possible values and pot size is discretized every 2 big blind. For example at the Flop stage, the table is composed of $5 \times 4 = 20$ values. Tables at other stages are bigger since the maximum size of the pot is bigger. Those tables have been generated by resorting to expert knowledge. They are not detailed in the paper for size reasons but are available at http://sequel.futurs.inria.fr/maitrepierre/basis-strategies-tables

At the beginning we only consider one strategy (tight/aggressive), and after several hands against an opponent, we are able to identify some weakness in our strategy, so we add new strategies. A new strategy is a convex combination of the two basis strategies. For example since the initial strategy is very *tight*, adding a *looser* strategy and selecting which one to use (by a method described in the next section) will improve the global behavior. Figure 3 shows the improvement of adding new strategies in games against Vexbot [6]. In the version which participate to the AAAI'07 we considered 5 different strategies built from the 2 basis strategies.

3.3 Strategy Selection

We have seen in the previous section that the different styles of play about the opponent yield different belief updates, which in turn defines different basic strategies. We now have to select a good one.

To do that we use a bandit algorithm called UCB (for *Upper Confidence Bounds*), see [2]. This algorithm allows us to find a good tradeoff between exploitation (use what is believed to be the best strategy) and exploration (select another apparently sub-optimal strategy in order to get additional information about the opponent). UCB algorithm works by defining a confidence bound for each possible strategy, and selects the strategy that has the highest upper bound. In our version we use a slightly modified version of the algorithm named *UCB-tuned* [1], which takes into account the empirical variance of the obtained rewards. For strategy i, the bound is defined as:

$$B_i(n) \stackrel{\text{def}}{=} \sigma_i \sqrt{\frac{2 \ln n}{n_i}}$$

where:

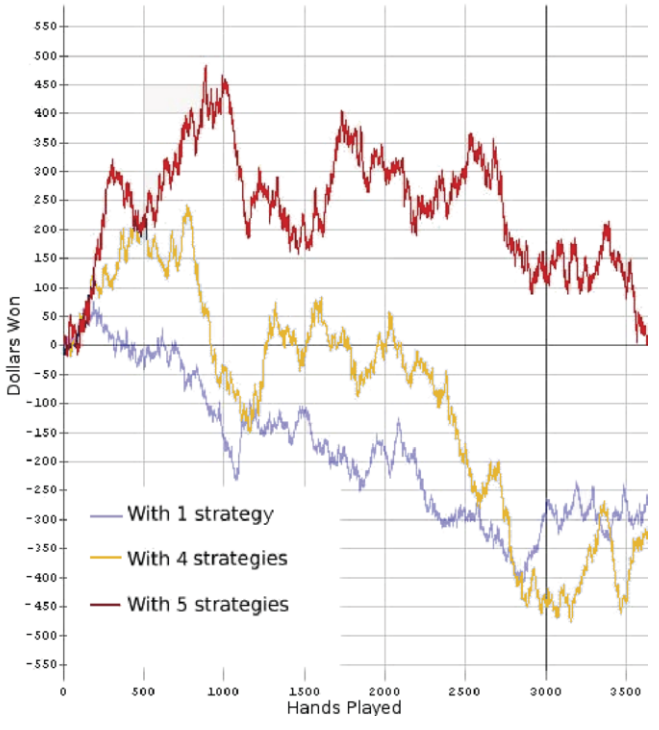

Figure 3. Performance of one, four, and five strategies against Vexbot (which is an adaptive bot). We observe that the resulting meta-strategy is stronger because it adapts automatically to the opponent and is less predictable.

- n is the number of hand played.
- n_i is the number of times strategy i was played.
- σ_i is the empirical standard deviation of the rewards.

The UCB (-tuned) algorithm consists in selecting the strategy i which has the highest upper bound:

$$\bar{x}_i(n) + B_i(n),$$

where $\bar{x}_i(n)$ is the average rewards won by strategy i up to time n.

This version of UCB assumes that the rewards corresponding to each strategy are independent and identically distributed samples of fixed random variables. However, in Poker, our opponent may change his style of play and search for a counter-strategy which adapts to ours. In order to detect possible changes in the opponent strategy, we combine the UCB selection policy to a change-point detection technique.

The change-point detection technique should detect brutal decrease in the rewards when using the best strategy (this would correspond to an adaptation of the opponent to our strategy). For this purpose, we define a lower bound on each strategy:

$$L_i(n) \stackrel{\text{def}}{=} \bar{x}_i(n) - B_i(n),$$

and we compute the moving average rewards, written $\bar{x}_i(n-200:n)$, on a window corresponding to the last 200 played hands with each strategy. We say that there is change-point detection if, for the current best strategy i, it happens that $\bar{x}_i(n-200:n) \leq L_i(n)$ (i.e. the average rewards obtained over a certain time period is actually worse than the current lower bound on the expected rewards), then we give the interpretation that this strategy is starting to be less effective against the opponent (the opponent adapts to it), and we decide to forget the period when strategy i was the best, and recompute the bounds and the average rewards for each strategy but only over the 200 lasts hands.

Change point detection is illustrated in Figure 4: near the 370^{th} hand, strategy 1 average income has decreased to be under the lower confidence bound, so we recompute new average and bounds.

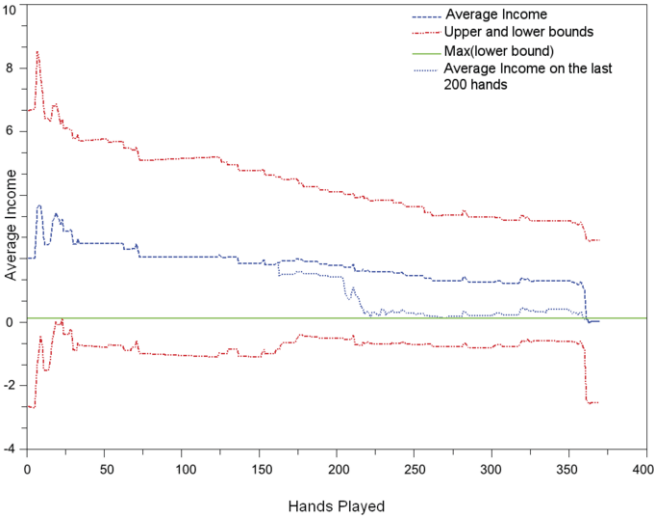

Figure 4. Change point detection. After hand 370 the average reward of strategy 1 goes under UCB's bound. So the historic of hands is reset and we recompute new bounds for each strategies.

4 Numerical results

We tested our bot against Sparbot [5], Vexbot [6] which are the current best bots in limit heads-up Poker, an AlwaysCall bot and an AlwaysRaise bot (2 deterministic bots which always play the same action). The tests was 1000 hands sessions and we test our bot against each bots on 10 sessions. Vexbot and our bot memory was reset after each session. Results are presented in Table 1.

	Our bot	Vexbot	Sparbot	AlwCall	AlwRaise
Our bot		+0.05	+0.02	+1.01	+1.87
Vexbot	-0.05		+0.056	+1.04	+2.98
Sparbot	-0.02	-0.056		+0.47	+1.34
AlwaysCall	-1.01	-1.04	-0.47		=0.00
AlwaysRaise	-1.87	-2.98	-1.34	=0.00	

Table 1. Matches against different Bots, over 10 sessions of 1000 hands. Results are expressed in smallBlind won per Hand for the line player versus column one.

We have studied UCB's behavior all along a match against Vexbot, studying this match seems more interesting to us since Vexbot is the only bot which adapts to his opponent's behavior.

Figure 5 shows the different uses of the strategies all over the match. We can see that some strategies are favored than others during

such or such periods: between hands 1500 and 2500, strategies 1 and 2 are very often used. After, strategies 3 and 4 are used over the 1000 following hands. This shows our opponent's capacity of adaptation and the fact that UCB, thanks to change point detection, detects this adaptation and changes the current strategy. We can see that strategy 5 isn't very used during the match but the addition of this strategy improves the performance of our bot figure 3 shows the performance difference before and after the add of strategy 5. We must keep in mind that UCB not only perform a choice over the strategies but also bring us a strategy which is a mix of the basic ones. So Vexbot defeats all our basic strategies but is defeated by the meta-strategy.

Also note that Sparbot which is a pseudo-equilibrium is defeated. That is something very interesting because equilibrium players, since they don't have any weakness to exploit, are a nightmare for adaptive play. It means that even taking no care of computing an equilibrium play on our basis strategies, the meta-strategy can adapt to be not so far of an equilibrium.

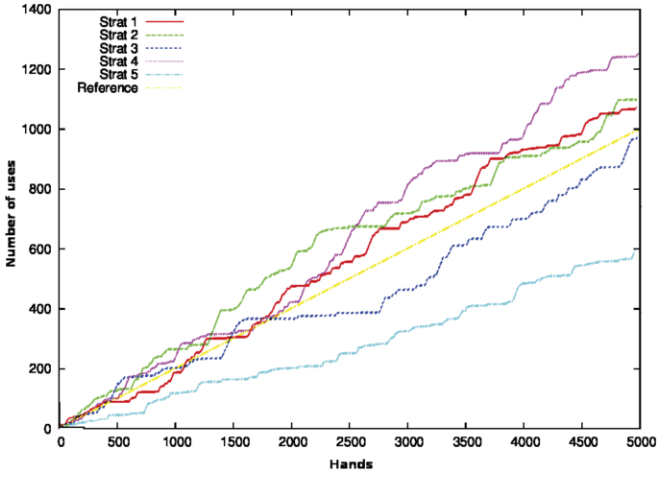

Figure 5. These curves show the number of uses of each strategy over hands played. Reference curve represents an uniformly use of each strategy. Plateaus represents periods during which a strategy isn't used whereas slopes show great use of it.

We register our bot to the AAAI'07 Computer Poker Competition. It takes part in two competitions, the online learning competition and the equilibrium one. Results can be viewed at http://www.cs.ualberta.ca/~pokert. Even if our approach is able to defeat all the AAAI'06 bots, we didn't perform very well in this competition (not in top 5 bots). We see several reasons to this: firstly, our approach require a lot a computer time during the match. So we had to limit the monte carlo exploration in order to comply with the time limit. Secondly, the strategy of the top competitors is really very close to a Nash equilibrium. So as our different strategies are not computed to be Nash equilibrium, our aggressive play is defeated. In fact, in future version we think that the meta-strategy obtained by uniformly choose one of our basis strategy should lead to a Nash equilibrium. Doing so will ensure us to not losing chips during the exploration stage because at the very beginning, UCB performs a near uniform exploration of all strategies. Moreover, it would offer a good response to a Nash equilibrium player: an other Nash equilibrium.

An other future improvement will be the update at the same time of the expectation of several arms of the UCB. This is possible because there is correlations between the rewards of each arm: if a slightly aggressive style doesn't work, a very aggressive one will probably fail too. It will allow us to add more basic strategies and having more subtle attempt of exploitation.

5 CONCLUSIONS

We presented an Texas Hold'em limit poker player which adapts its playing style to his opponents. It combines beliefs update methods to obtain different strategies. The use of UCB algorithm enables a fast adaptation to modifications of opponent's playing style. For humans player, the produced bot seems more pleasant than equilibriums ones since it tries different strategies against his opponent. Moreover due to the UCB selection, the style of play varies very quickly which sometimes give the illusion that the computer tried to trap the opponent.

Using different strategies and choosing the right one depending on opponents playing styles seems to be promising idea and should be adapted to multi-player gaming.

REFERENCES

[1] J.Y. Audibert, R. Munos, and C. Szepesvári. Use of variance estimation in the multi-armed bandit problem. NIPS Workshop on on-line trading of exploration and exploitation, Vancouver, 2006.
[2] Peter Auer, Nicolò Cesa-Bianchi, and Paul Fischer, 'Finite-time analysis of the multiarmed bandit problem', *Machine Learning*, **47**(2/3), 235–256, (2002).
[3] D. Billings, 'The first international roshambo programming competition', *The International Computer Games Association Journal*, **1**(23), 42–50, (2000).
[4] D. Billings, 'Thoughts on roshambo', *The International Computer Games Association Journal*, **1**(23), 3–8, (2000).
[5] D. Billings, N. Burch, A. Davidson, R. Holte, J. Schaeffer, T. Schauenberg, and D. Szafron. Approximating game-theoretic optimal strategies for full-scale poker, 2003.
[6] D. Billings, A. Davidson, T. Schauenberg, N. Burch, M. Bowling, R. Holte, J. Schaeffer, and D. Szafron, 'Game-tree search with adaptation in stochastic imperfect-information games', *Computers and Games: 4th International Conference*, 21–34, (2004).
[7] Darse Billings, Aaron Davidson, Jonathan Schaeffer, and Duane Szafron, 'The challenge of poker', *Artificial Intelligence*, **134**(1-2), 201–240, (2002).
[8] Darse Billings, Lourdes Pena, Jonathan Schaeffer, and Duane Szafron, 'Using probabilistic knowledge and simulation to play poker', in *AAAI/IAAI*, pp. 697–703, (1999).
[9] Doyle Brunson, *Super System: A Course in Power Poker,*, Cardoza Publishing, 1979.
[10] Murray Campbell, A. Joseph Hoane Jr., and Feng hsiung Hsu, 'Deep blue', *Artificial Intelligence*, **134**, 57–83, (2002).
[11] Andrew Gilpin and Tuomas Sandholm, 'Finding equilibria in large sequential games of imperfect information', in *EC '06: Proceedings of the 7th ACM conference on Electronic commerce*, pp. 160–169, New York, NY, USA, (2006). ACM.
[12] Andrew Gilpin and Tuomas Sandholm, 'Better automated abstraction techniques for imperfect information games, with application to texas hold'em poker', in *AAMAS '07: Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems*, pp. 1–8, New York, NY, USA, (2007). ACM.
[13] J. F. Nash. Equilibrium points in n-person games, 1950.
[14] J. Schaeffer and R. Lake. Solving the game of checkers, 1996.
[15] A. Selby. Optimal strategy for heads-up limit holdem.
[16] Martin Zinkevich, Michael Johanson, Michael Bowling, and Carmelo Piccione, 'Regret minimization in games with incomplete information', in *Advances in Neural Information Processing Systems 20*, eds., J.C. Platt, D. Koller, Y. Singer, and S. Roweis, MIT Press, Cambridge, MA, (2008).