Monte-Carlo Tree Search in Poker Using Expected Reward Distributions

Conference Paper · November 2009

DOI: 10.1007/978-3-642-05224-8_28 · Source: DBLP

3 authors, including:

Kurt Driessens
Maastricht University
75 PUBLICATIONS **1,757** CITATIONS
SEE PROFILE

Jan Ramon
KU Leuven
149 PUBLICATIONS **3,084** CITATIONS
SEE PROFILE

Monte-Carlo Tree Search in Poker using Expected Reward Distributions

Guy Van den Broeck, Kurt Driessens, and Jan Ramon

Department of Computer Science, Katholieke Universiteit Leuven, Belgium
{guy.vandenbroeck,kurt.driessens,jan.ramon}@cs.kuleuven.be

Abstract. We investigate the use of Monte-Carlo Tree Search (MCTS) within the field of computer Poker, more specifically No-Limit Texas Hold'em. The hidden information in Poker results in so called MIXIMAX game trees where opponent decision nodes have to be modeled as chance nodes. The probability distribution in these nodes is modeled by an opponent model that predicts the actions of the opponents. We propose a modification of the standard MCTS selection and backpropagation strategies that explicitly model and exploit the uncertainty of sampled expected values. The new strategies are evaluated as a part of a complete Poker bot that is, to the best of our knowledge, the first exploiting no-limit Texas Hold'em bot that can play at a reasonable level in games of more than two players.

1 Introduction

Poker playing computer bots can be divided into two categories. There are the *game-theoretic bots*, that play according to a strategy that gives rise to an approximate Nash equilibrium. These bots are difficult, or impossible to beat, but are also not able to exploit possible non-optimalities in their opponents. Game theoretic strategy development also becomes very complex as the number of opponents increases, so most bots that try to play Nash equilibria, do so in a heads-up setting, i.e., with only one opponent. The other type of bot is the *exploiting bot* that employs game tree search and opponent modeling techniques to discover and exploit possible weaknesses in opponents. In this paper, we focus on the second type of bot.

Research in computer Poker has been mainly dealing with the *limit* variant of Texas Hold'em. In limit Poker, bet sizes are fixed as well as the total number of times a player can raise the size of the pot. Only recently [1], attention has started shifting to the no-limit variant and the increased complexity that it brings. In fact, all exploiting bots developed so far, deal with the heads-up limit version of the game. The limit game tree consists of about 10^{18} nodes and modern bots are capable of traversing the complete tree in search of optimized actions through a full-width depth-first search [2]. Gilpin et al.[1] estimate the game tree in heads-up no-limit Texas Hold'em (with bet-size discretization) to reach a node-count of about 10^{71}.

The true tournament version of Texas Hold'em is played with a maximum of 10 players per table, expanding the size of the game tree even more. Traversing the whole game tree is in this case not an option. While existing bots can either play limit or heads-up poker, the bot we describe here can deal with the full complexity of the poker game by employing sampling methods and bet-size discretization.

Monte-Carlo Tree Search (MCTS) is a best-first search technique that estimates game tree node values based on the results from simulated gameplay. Its main goal is to replace exhaustive search through the game tree with well founded sampling methods. The most popular heuristic for the best-first selection in MCTS is UCT [3], which stands for Upper Confidence bounds applied to Trees. UCT is a translation of the upper confidence bound selection algorithms developed for bandit-based problem that puts a limit on the regret caused by exploration moves. In this paper we suggest the use of MCTS methods in no-limit Texas Hold'em Poker to limit the extent of the tree the Poker bot has to search, thereby reducing the computational requirements to reach a reasonable play-level.

As a first contribution, we apply MCTS in the context of MIXIMAX game trees [2] as they are encountered in incomplete information games such as Poker where opponent actions can not simply be predicted as value minimization actions. While adaptations of MCTS have been defined for EXPECTIMAX game trees [4] that deal with non-determinism such as dice rolls, we are not aware of any previous work on MCTS in incomplete information games. As a second contribution, we adapt both the backpropagation and selection steps in MCTS to incorporate explicit sample variance information in the game tree nodes.

The rest of the paper is structured as follows. The following section briefly presents no-limit Texas Hold'em Poker and the game tree it results in. Section 3 discusses MCTS and some standard implementation choices for it. In section 4 we propose extensions of MCTS that deal explicitly with the sampling distribution in each node. The proposed strategies are evaluated in section 5 after which we conclude.

2 The Poker game tree

Texas Hold'em Poker is played by two or more people who each hold two hidden cards. The game starts with a betting round where people can invest money in a shared pot by putting in a bet. Alternatively, they can check and let the next player make the first bet. Players must match every bet made by investing an equal amount (calling) or by investing even more (raising). When a player fails to do so and gives up, he folds. The betting round ends when everybody has folded or called. To make sure each game is played, the first two players in the first round are forced to place bets, called small and big blinds. The big blind is double the amount of chips of the small blind.

Next, a set of three community cards (called the Flop) is dealt to the table, followed by a new betting round. These cards are visible to every player and

the goal is to combine them with the two hidden cards to form a combination of cards (also called a hand). One card is added to the set of community cards twice more (the first one called the Turn and the last called the River) to make a total of five, each time initiating a new betting round. Finally, the player with the best hand that hasn't folded wins the game and takes the pot.

From a game tree perspective, this gives rise to the following node types:

Leaf nodes: Evaluation of the expected value of the game in these nodes is usually trivial. In Poker, we choose to employ the expected sum of money a player has after the game is resolved. In case the player loses, this is the amount of money he didn't bet. In case the player wins, it is that amount plus the current pot, with a possible distribution of the pot if multiple players have similar poker hands. Leafs can also take probability distributions over currently unknown cards into account.

Decision nodes: These are the nodes where the bot itself is in control of the game. In non-deterministic, fully observable games, these would be the nodes where the child that maximizes the expected outcome is selected. While in Poker, it can be beneficial to not always select the optimal action and remain somewhat unpredictable, they should still be treated as maximization nodes during the game tree search.

Chance nodes: These nodes are comparable to those encountered in EXPECTIMAX games. In Poker, they occur when new cards are dealt to the table. In these nodes, children are selected according to the probability distribution connected to the stochastic process steering the game at these points.

Opponent nodes: In complete information games, the opponent will try to minimize the value in the nodes where he can make a choice. However, in poker, we do not know the evaluation function of the opponent, which will typically be different because the opponent has different information. Neither does the opponent know our evaluation function. Instead of treating the opponent nodes as min-nodes one can therefore consider these nodes as chance nodes. The big difference is that the probability distribution in opponent nodes is unknown and often not static. The probability distribution over the different options the opponent can select from can be represented (and learned) as an opponent-model.[1]

These nodes make up a MIXIMAX game tree that models the expected value of each game state. Because this tree is too big, it is impossible to construct. We can only approximate the expected values using an incomplete search procedure like MCTS.

The opponent model A full study or overview of possible factorizations of the opponent model in Poker is out of the scope of this paper. However, since the performance of the MCTS bots as described in this paper is greatly dependent

[1] Depending on the expressivity of the opponent-model, this approach can either try to model the hidden information in the game or reduce it to the EXPECTIMAX model.

on it, we include the employed opponent modeling strategy for completeness reasons.

Assume that the action performed in step i is represented as A_i, that the community cards dealt at step i are represented as C_i and that H represents the hand cards held by all active players. We use an opponent model factored into the two following distributions:

1. $P(A_i|A_0 \dots A_{i-1}, C_0 \dots C_i)$: This model predicts the actions of all players, taking previous actions and dealt community cards into account.
2. $P(H|A_0 \dots A_n, C_0 \dots C_n)$: This model predicts the hand cards of all active players at showdown (represented as step n).

The first probability can be easily estimated using observations made during the game, as all relevant variables can be observed. This is only possible because we leave hand cards out of the first model. Predicting the exact amount of a bet or a raise is difficult. We decided to deal with this by treating minimal bets and all-in bets (i.e. when a player bets his entire stack of money) as separate cases and discretizing the other values.

In the end we need to know who won the game and therefore, we need to model the hand cards with the second probability. This one can be harder to model as "mucking" (i.e. throwing away cards without showing them to other players) can hide information about the cards of players in a loosing position. To make abstraction of the high number of possible hand-card combinations, we reduce the possible outcomes to the rank of the best hand that the cards of each active player result in [5] and again discretize these ranks into a number of bins. This approach has the advantage that mucked hands still hold some information and can be attributed to bins with ranks lower than that of the current winning hand.

The actual models are learned using the Weka toolkit [6] from a dataset of a million games played in an online casino. Each action in the dataset is an instance to learn the first model from. Each showdown is used to learn the second model. The game state at the time of an action or showdown is represented by a number of attributes. These attributes include information about the current game, such as the round, the pot and the stack size of the player involved. It also includes statistics of the player from previous games, such as the bet and fold frequencies for each round and information about his opponents, such as the average raise frequency of the active players at the table. With these attributes, Weka's M5P algorithm [7] was used to learn a regression tree that predicts the probabilities of each action or hand rank bin at showdown.

The use of features that represent windowing[2] statistics about previous games of a player in a model learned on a non-player-specific data set has the advantage of adapting the opponent model to the current opponent(s). It is important to note that there are usually not sufficient examples from a single individual player to learn something significant. We circumvent this problem by learning what is

[2] Statistics are kept over a limited number of past games to adapt to changing game circumstances.

common for all players from a large database of games. At the same time, to exploit player-specific behavior, we will learn a function from the game situation and player-specific statistics to a prediction of his action. A key issue here is to select player-specific statistics which are easy to collect with only a few observed games, while capturing as much as possible about the players' behavior.

3 Monte-Carlo tree search

Monte-Carlo Tree Search is a best-first search strategy. It revolutionized research in computer-Go and gave rise to Go-bots such as CrazyStone [8], MoGo [9] and Mango [10], which represent the current top-ranks in computer Go competitions. The original goal of MCTS was to eliminate the need to search minimax game trees exhaustively and sample from the tree instead. Later, extensions of the MCTS technique were formulated that could deal with expectimax trees such as those encountered in backgammon [11]. To the best of our knowledge, MTCS has not yet been adapted to deal with miximax trees that are encountered in incomplete information games such as Poker.

MCTS incrementally builds a subtree of the entire game tree in memory. For each stored node P, it also stores an estimate $\hat{V}(P)$ of the expected value $V^*(P) = \mathbb{E}[r(P)]$ of the reward $r(P)$ of that node together with a counter $\mathrm{T}(P)$ that stores the number of sampled games that gave rise to the estimate. The algorithms starts with only the root of the tree and repeats the following 4 steps until it runs out of computation time:

Selection: Starting from the root, the algorithm selects in each stored node the branch it wants to explore further until it reaches a stored leaf (This is not necessarily a leaf of the game tree). The selection strategy is a parameter of the MCTS approach.

Expansion: One (or more) leafs are added to the stored tree as child(ren) of the leaf reached in the previous step.

Simulation: A sample game starting from the added leaf is played (using a simple and fast game-playing strategy) until conclusion. The value of the reached result (i.e. of the reached game tree leaf) is recorded. MCTS does not require an evaluation heuristic, as each game is simulated to conclusion.

Backpropagation: The estimates of the expected values $V^*(P) = \mathbb{E}[r(P)]$ (and selection counter $\mathrm{T}(P)$) of each recorded node P on the explored path is updated according to the recorded result. The backpropagation strategy is also a parameter of the MCTS approach.

After a number of iterations of the previous four steps, an action-selection strategy is responsible for choosing a good action to be executed based on the expected value estimates and the selection counter stored in each of the root's children. It has been suggested to pick the child with the highest expected value or the highest visit count. Our experiments show that the choice of action-selection strategy has no significant impact.

3.1 UCT sample selection

Node selection or sample selection as described above is quite similar to the widely studied Multi-Armed Bandit (MAB) problem. In this problem, the goal is to minimize regret[3] in a selection task with K options, where each selection c_i results in a return according to a fixed probability distribution $r(c_i)$. The **Upper Confidence Bound** selection strategy (UCB1) is based on the Chernoff-Hoeffding limit that constrains the difference between the sum of random variables and the expected value. For every option c_i, UCB1 keeps track of the average returned reward $\mathbb{E}[r(c_i)]$ as well as the number of trials $\mathrm{T}(c_i)$. After sampling all options once, it selects the option that maximizes:

$$\hat{V}(c_i) + C\sqrt{\frac{\ln \mathrm{T}(P)}{\mathrm{T}(c_i)}} \tag{1}$$

where $\mathrm{T}(P) = \sum_j \mathrm{T}(c_j)$ is the total number of trials made. In this equation, the average reward term is responsible for the exploitation part of the selection strategy, while the second term, which represents an estimate of the upper bound of the confidence interval on $\mathbb{E}[r(c_j)]$, takes cares of exploration. C is a parameter that allows tuning of this exploration-exploitation trade-off. This selection strategy limits the growth rate of the total regret to be logarithmic in the number of trials [12].

UCB applied to Trees (UCT) [3] extends this selection strategy to Markov decision processes and game trees. It considers each node selection step in MCTS as an individual MAB problem. Often, UCT is only applied after each node was selected for a minimal number of trials. Before this number is reached, a predefined selection probability is used. UCT assumes that all returned results of an option are independently and identically distributed, and thus that all distributions are stationary. For MCTS, this is however not the case, as each sample will alter both $\hat{V}(s)$ and $\mathrm{T}(s)$ somewhere in the tree and thereby also the sampling distribution for following trials. Also, while the goal of a MAB problem is to select the best option as often as possible, the goal of the sample selection strategy in MCTS is to sample the options such that the best option can be selected at the root of the tree in the end. While both goals are similar, they are not identical. Nonetheless, the UCT heuristic performs quite well in practice.

3.2 Sample selection in CrazyStone

The CrazyStone selection strategy [8] chooses options according to the probability that they will return a better value than the child with the highest expected value. Besides the estimated expected value $\mathbb{E}[r(P)]$ each node P also stores the standard deviation of the estimates of the expected value $\sigma_{\hat{V},P}$.

[3] Regret in a selection task is the difference in cumulative returns compared to the return that could be attained when using the optimal strategy.

The probability of selecting an option c_i is set to be proportional to:

$$\mathrm{P}(c_i) \sim \exp\left(-2.4 \frac{\hat{V}(c_{best}) - \hat{V}(c_i)}{\sqrt{2(\overline{\sigma}(c_{best})^2 + \overline{\sigma}(c_i)^2)}}\right). \tag{2}$$

where c_{best} is the option with the highest expected value. Under the assumption that values follow a Gaussian distribution, this formula approximates the target probability.

The CrazyStone-selection is only applicable for deterministic minimax-trees. Non deterministic nodes cause each node to have a probability distribution over expected rewards that does not converge to a single value. This stops $\overline{\sigma}(n)$ from converging to 0 and causes the number of samples using non-optimal options to remain non-negligible.

3.3 Backpropagation in mixed nodes

For both chance and opponent nodes the expected value of child-nodes are given a weight proportional to the amount of samples used:

$$\hat{V}(n) = \sum_j \frac{\mathrm{T}(c_j)\hat{V}(c_j)}{\mathrm{T}(n)} \tag{3}$$

This is simply the average of all values sampled through the current node.

3.4 Backpropagation in decision nodes

In decision nodes, we have a choice how to propagate expected values. The only requirement is that the propagated value converges to the maximal expected value as the number of samples rises. Two simple strategies present themselves:

- Use the average sampling result as it is used in mixed nodes: when using the correct selection strategy, for example UCT, the majority of samples will eventually stem from the child with the highest expected value and this will force equation 3 to converge to the intended value. Before convergence, this strategy will usually underestimate the true expected value, as results from non-optimal children are also included.
- Use the maximum of the estimates of the expected values of all child nodes:

$$\hat{V}(P) = \max_j \hat{V}(c_j). \tag{4}$$

  As, in the limit, $\hat{V}(P)$ will converge to the true value $V^*(P)$, the maximum of all estimates will also converge to the correct maximum. This approach will usually overestimate the true expected value as any noise present on the sampling means is also maximized.

4 Sample selection and backpropagation strategy

In this section, we introduce new sample-selection and backpropagation strategies for MCTS that take advantage of information about the probability distribution over the sampling results. Existing techniques do not take fully into account the consequences of having non-deterministic nodes in the search tree. The CrazyStone algorithm does use the variance of the sample in its nodes, but its strategy only works well for deterministic games.

Not only backpropagating an estimate of the value of a node but also a measure of the uncertainty on this estimate has several advantages. First, as one can see in our derivations below, it allows a more accurate estimation of the expected value of nodes higher in the tree based on its children. Second, better sample selection strategies can be used. In particular, instead of the Chernoff-Hoeffding based heuristic from Equation (1), one can use an upper bound of the confidence interval taking the variance into account, and one can select for sampling the child node with highest

$$\hat{V}(c_i) + C.\sigma_{\hat{V},c_i} \tag{5}$$

where C is a constant and $\sigma_{\hat{V},c_i}$ is the standard error on $\hat{V}$. We call this strategy UCT+. Here, the first term is again the exploitation term and the second term an exploration term. In contrast to Equation (1) it takes into account the uncertainty based on the actually observed samples of the child node.

Let P be a node of the search tree. We will denote with $V^*(P)$ the value of P, i.e. the expected value of the gain of the player under perfect play. Unfortunately, we do not know yet these values, nor the optimal strategies that will achieve them. Assume therefore that we have estimates $\hat{V}(P)$ of the $V^*(P)$. It is important to know how accurate our estimates are. Therefore, let

$$\sigma_{\hat{V},P}{}^2 = \mathbb{E}[(\hat{V}(P) - V^*(P))^2].$$

For example, if P is a leaf node, and if we have J samples $x_1 \dots x_J$ of the value of P, then we can estimate

$$\hat{V}(P) = \frac{1}{J}\sum_{j=1}^{J} x_j$$

The unbiased estimate of the population variance is

$$\frac{1}{J-1}\sum_{j=1}^{J}(\hat{V}(P) - x_j)^2.$$

As $\hat{V}(P)$ is the average of J samples of this population, we can estimate the variance on it (the standard error) with

$$\sigma_{\hat{V},P}{}^2 = \mathbb{E}[(\hat{V}(P) - V^*(P))^2] = \frac{1}{J(J-1)}\sum_{j=1}^{J}(\hat{V}(P) - x_j)^2$$

For large samples, the estimate $\hat{V}(P)$ converges to $V^*(P)$.

Suppose now that P is a node of the search tree and that it has n children $c_1 \dots c_n$. Given $\hat{V}(c_i)$ and $\sigma_{\hat{V},i}$ for $i = 1..n$, we are interested in finding an estimate $\hat{V}(P)$ for the value $V^*(P)$ of the parent node P and a variance $\sigma_{\hat{V},P}$ on this estimate. In Section 4.1 we will consider this problem for decision nodes, and in Section 4.2 we will discuss opponent nodes and chance nodes.

4.1 Decision nodes

Let P be a decision node and let c_i $(i = 1 \dots n)$ be its children. We will here assume that when the game would ever get into node P, we will get extra time to decide which of the children c_i, $i = 1 \dots n$ is really the best choice. Let

$$e_i = \hat{V}(c_i) - V^*(c_i)$$

be the error on the estimate of the value of child c_i. We have

$$\hat{V}(P) = \int_{e_1 \dots e_n} (\max_{i=1}^{n}(\hat{V}(c_i) - e_i))\, \mathrm{P}(e_1 \dots e_n) \mathrm{d}e_1 \dots \mathrm{d}e_n$$

If the estimates $\hat{V}(c_i)$ are obtained by independent sampling, then the errors e_i are independent and we can write

$$\hat{V}(P) = \int_{e_1 \dots e_n} (\max_{i=1}^{n}(\hat{V}(c_i) - e_i)) \prod_{i=1}^{n} \mathrm{P}(e_i) \mathrm{d}e_1 \dots \mathrm{d}e_n$$

For the variance of $\hat{V}(P)$ we can write

$$\sigma_{\hat{V},P} = \int_{e_1 \dots e_n} (\hat{V}(P) - \max_{i=1}^{n}(\hat{V}(c_i) - e_i))^2 \prod_{i=1}^{n} \mathrm{P}(e_i) \mathrm{d}e_1 \dots \mathrm{d}e_n$$

If we assume the errors e_i are normally distributed, these integral can be evaluated symbolically, at least if the number of children is not too high. Unfortunately, as the maximum of several normal distributions is not a normal distribution itself, we can not expect such assumption to be correct. However, we can obtain a relatively close and efficiently computable approximation, by ignoring this problem and acting as if all errors would be normally distributed.

4.2 Opponent nodes and chance nodes

As discussed in Section 2 chance nodes and opponent nodes can be treated equivalently, using an opponent model to dictate the probability distribution in opponent nodes. Let P be a chance node. Let c_i, $i = 1 \dots n$ be the children of P and let their probability be p_i. Then, we have

$$\hat{V}(P) = \sum_{i=1}^{n} p_i \hat{V}(c_i)$$

and

$$\sigma^2_{\hat{V},P} = \sum_{i=1}^{n} p_i \sigma^2_{\hat{V},i}$$

Both the total expected value and its standard error are weighed by p_i. In chance nodes, p_i describes the stochastic process. In opponent nodes, the probability is provided by the opponent model.

5 Experiments

In this section we will evaluate our new algorithm empirically. In particular, we will investigate the following questions:

Q1 : Is the new algorithm able to exploit weaknesses in rule-based players?
Q2 : How does the proposed backpropagation algorithm compare to existing algorithms?
Q3 : What is the effect of available computing resources (thinking time)?
Q4 : What is the effect of the proposed sample selection algorithm?

5.1 Setup

Ideally, we would let our bots play against strong existing bots. Unfortunately, the few no-limit bots that exist are not freely available. Such a comparson would not really be fair as existing bots only play heads-up poker, whereas our bot is more generally applicable. Because it is the first exploiting no-limit bot, a comparison of exploitative behaviour would also be impossible.

We implemented the following poker playing bots to experiment with:

- StdBot, the standard MCTS algorithm.
- NewBot, our new algorithm as explained in the section 4.
- RuleBotHandOnly, a naive rule-based bot. His behaviour only depends on the hand cards he's dealt.
- RuleBotBestHand, a more complex rule-based bot. He estimates the probability that he has the strongest hands and bases his strategy entirely on that information.

StdBot and NewBot both use the opponent model that's described in Section 2. RuleBotHandOnly and RuleBotBestHand don't use any opponent model. In our experiments, we each time let the bots (with certain parameters) play a large number of games against each other. Each player starts every game with a stack of 200 small bets (sb). The contant C in Equation (1) is always equal to 50 000. To reduce the variance in the experiments, players are dealt each other's cards in consecutive games. We compute the average gain of a bot per game in small bets.

5.2 Exploitative behaviour (Q1)

We let NewBot play against RuleBotHandOnly and RuleBotBestHand in a large number of 3-player games. Figure 1 shows the average profit of each bot. The dotted lines are the σ-confidence interval around the estimated average profit.

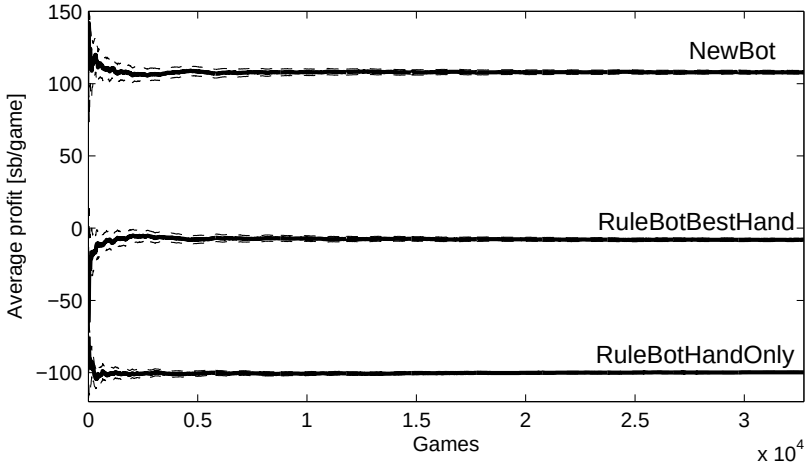

Fig. 1. Average profit of NewBot against two rule-based bots.

NewBot is clearly superior to both rule-based bots. The more complex RuleBotBestHand is clearly superior to RuleBotHandOnly. This experiment shows that MCTS in combination with an opponent model is capable of strong exploitative behaviour and that exploitative game tree search bots can be developed for no-limit poker with more than 2 players.

5.3 Comparison with standard MCTS (Q2)

To test the new backpropagation algorithm, we let NewBot that uses the maximum distribution backpropagation strategy play against StdBot that uses the standard backpropagation algorithm, where a sample-weighted average of the expected value of the children is propagated. To avoid overestimation, the maximum distribution only replaces the standard algorithm if a node has been sampled 200 times. Both bots were allowed 1 second computation time for each decision and were otherwise configured with the same settings. Figure 2 shows the average profit of both bots.

The backpropagation algorithm we propose is significantly superior in this experiment. It wins with 3 small bets per game, which would be a very large profit among human players. Using other settings for the UCT selection algorithm in

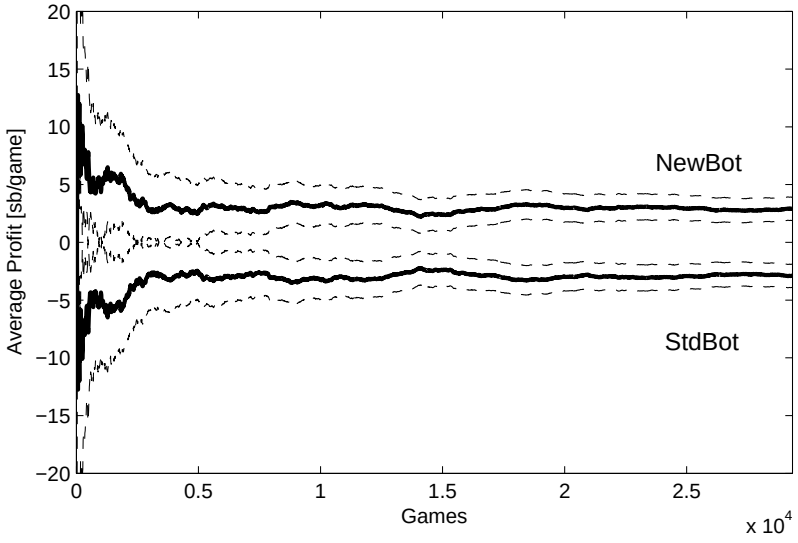

Fig. 2. Average profit of NewBot vs. StdBot

other experiments, the results were not always definitive and sometimes the proposed strategy even performed slightly worse that the standard algorithm. Therefore, it is hard to draw general conclusions for question Q2 based on these experiments. Further parameter tuning for both approaches is necessary before any strong claims are possible either way.

5.4 Influence of thinking time (Q3)

In this context, we investigated the influence of available computation time on the effectiveness of the maximum distribution backpropagation algorithm. Figure 3 shows the average profit of NewBot in heads-up games against StdBot for different computation times.

Even though there are some outliers, the balance is cleary in favour of NewBot for longer thinking times. If enough time is available, the advantages of the maximum distribution strategy start to outweigh the disadvantage of being able to sample less.

We expect that, when more elaborate opponent modeling techniques are used that require a proportionally larger amount of time to be evaluated and thereby cause both techniques to sample a comparable number of game outcomes, the scale will also tip further in favor of the new backpropagation algorithm.

5.5 Influence of the proposed sample selection algorithm (Q4)

To investigate the influence of the proposed selection algorithm, we let a NewBot with the standard UCT algorithm using Equation (1) play against a New-

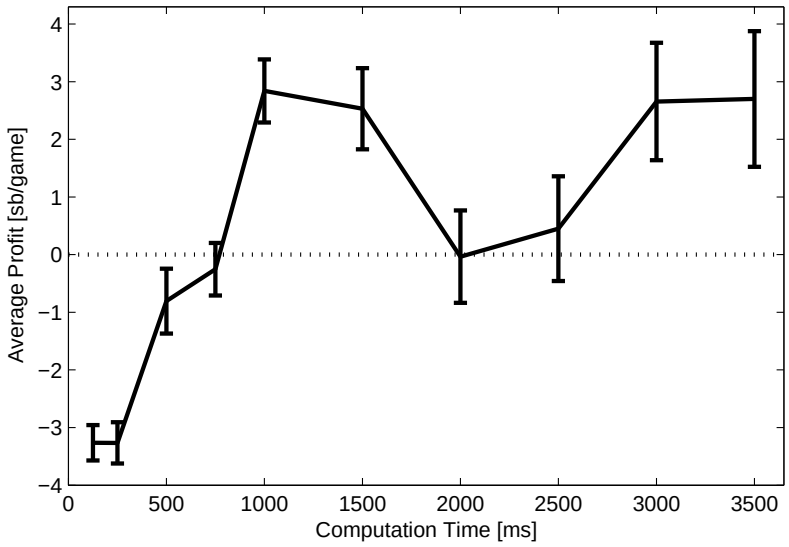

Fig. 3. Average profit of NewBot against StdBot depending on the calculation time.

Bot using Equation (5) for sampling with C equal to 2. Both bots are allowed to sample the game tree 25 000 times. All else being equal, the new UCT+ sample selection strategy clearly outperforms the standard UCT. As for Q2, not all parameter settings gave conclusive results and more parameter tuning is needed.

5.6 Experiments against human players

To estimate the bot's strength we performed tentative experiments with human players. NewBot was able to beat a human novice when playing 400 games with a significant profit of 8 sb/game. The same experiment against an experienced human player ended in a draw. From these initial experiments, we conclude that the bot plays at a reasonable level.

Large scale experiments against human players are planned for the future. We will integrate NewBot with a free online casino to simultaneously play hundreds of games. This will allow us to optimize the parameters of the search procedure for games against humans.

6 Conclusions

We've introduced MCTS in the context of Texas Hold'em Poker and the miximax game trees it results in. Using a new backpropagation strategy that explicitly models sample distributions and a new sample selection strategy, we developed the first exploiting multi-player bots for no-limit Texas Hold'em. In a number of

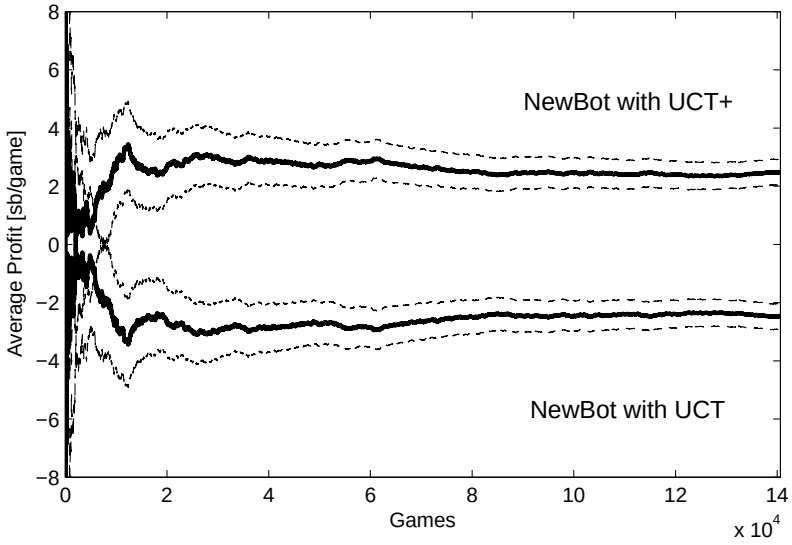

Fig. 4. Average profit of NewBot using UCT+ against NewBot using standard UCT.

experimental evaluations, we studied some of the conditions that allow the new strategies to result in a measurable advantage. An increase in available computation time translates in additional profit for the new approach. We expect a similar trend as the complexity of the opponent model surpasses the complexity of the backpropagation algorithm. In future work, we want to explore several directions. First, we want to better integrate the opponent model with the search. Second, we want to find a good way to deal with the opponent nodes as minimization nodes as well as non-deterministic nodes. Finally, we want to shift the goal of the search from finding the optimal deterministic action to finding the optimal randomized action (taking into account the fact that information is hidden).

Acknowledgements

Jan Ramon and Kurt Driessens are post-doctoral fellows of the Fund for Scientific Research (FWO) of Flanders.

References

1. Gilpin, A., Sandholm, T., Sørensen, T.: A heads-up no-limit Texas Hold'em poker player: discretized betting models and automatically generated equilibrium-finding programs. In: Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems-Volume 2, International Foundation for Autonomous Agents and Multiagent Systems Richland, SC (2008) 911–918

2. Billings, D.: Algorithms and assessment in computer poker. PhD thesis, Edmonton, Alta., Canada (2006)
3. Kocsis, L., Szepesvari, C.: Bandit based monte-carlo planning. Lecture Notes in Computer Science **4212** (2006) 282
4. Russell, S., Norvig, P.: Artificial intelligence: a modern approach. Prentice Hall (2003)
5. Suffecool, K.: Cactus kev's poker hand evaluator. http://www.suffecool.net/poker/evaluator.html (July 2007)
6. Witten Ian, H., Eibe, F.: Data Mining: Practical machine learning tools and techniques. Morgan Kaufmann, San Francisco (2005)
7. Wang, Y., Witten, I.: Induction of model trees for predicting continuous classes. (1996)
8. Coulom, R.: Efficient selectivity and backup operators in Monte-Carlo tree search. Lecture Notes in Computer Science **4630** (2007) 72
9. Gelly, S., Wang, Y.: Exploration exploitation in go: UCT for Monte-Carlo go. In: Twentieth Annual Conference on Neural Information Processing Systems (NIPS 2006). (2006)
10. Chaslot, G., Winands, M., Herik, H., Uiterwijk, J., Bouzy, B.: Progressive strategies for monte-carlo tree search. New Mathematics and Natural Computation **4**(3) (2008) 343
11. Van Lishout, F., Chaslot, G., Uiterwijk, J.: Monte-Carlo Tree Search in Backgammon. Computer Games Workshop (2007) 175–184
12. Auer, P., Cesa-Bianchi, N., Fischer, P.: Finite-time analysis of the multiarmed bandit problem. Machine Learning **47**(2-3) (2002) 235–256