Bandit Based Monte-Carlo Planning

Conference Paper *in* Lecture Notes in Computer Science · September 2006

DOI: 10.1007/11871842_29 · Source: DBLP

2 authors:

Levente Kocsis
Hungarian Academy of Sciences
28 PUBLICATIONS **2,016** CITATIONS
SEE PROFILE

Csaba Szepesvári
University of Alberta
307 PUBLICATIONS **8,992** CITATIONS
SEE PROFILE

Bandit based Monte-Carlo Planning

Levente Kocsis and Csaba Szepesvári

Computer and Automation Research Institute of the
Hungarian Academy of Sciences, Kende u. 13-17, 1111 Budapest, Hungary
kocsis@sztaki.hu

Abstract. For large state-space Markovian Decision Problems Monte-Carlo planning is one of the few viable approaches to find near-optimal solutions. In this paper we introduce a new algorithm, UCT, that applies bandit ideas to guide Monte-Carlo planning. In finite-horizon or discounted MDPs the algorithm is shown to be consistent and finite sample bounds are derived on the estimation error due to sampling. Experimental results show that in several domains, UCT is significantly more efficient than its alternatives.

1 Introduction

Consider the problem of finding a near optimal action in large state-space Markovian Decision Problems (MDPs) under the assumption a generative model of the MDP is available. One of the few viable approaches is to carry out sampling based lookahead search, as proposed by Kearns et al. [8], whose sparse lookahead search procedure builds a tree with its nodes labelled by either states or state-action pairs in an alternating manner, and the root corresponding to the initial state from where planning is initiated. Each node labelled by a state is followed in the tree by a fixed number of nodes associated with the actions available at that state, whilst each corresponding state-action labelled node is followed by a fixed number of state-labelled nodes sampled using the generative model of the MDP. During sampling, the sampled rewards are stored with the edges connecting state-action nodes and state nodes. The tree is built in a stage-wise manner, from the root to the leafs. Its depth is fixed. The computation of the values of the actions at the initial state happens from the leafs by propagating the values up in the tree: The value of a state-action labelled node is computed based on the average of the sum of the rewards along the edges originating at the node and the values at the corresponding successor nodes, whilst the value of a state node is computed by taking the maximum of the values of its children. Kearns et al. showed that in order to find an action at the initial state whose value is within the ϵ-vicinity of that of the best, for discounted MPDs with discount factor $0 < \gamma < 1$, K actions and uniformly bounded rewards, regardless of the size of the state-space fixed size trees suffice [8]. In particular, the depth of the tree is proportional to $1/(1-\gamma)\log(1/(\epsilon(1-\gamma)))$, whilst its width is proportional to $K/(\epsilon(1-\gamma))$.

Although this result looks promising,[1] in practice, the amount of work needed to compute just a single almost-optimal action at a given state can be overwhelmingly large. In this paper we are interested in improving the performance of this vanilla Monte-Carlo planning algorithm. In particular, we are interested in Monte-Carlo planning algorithms with two important characteristics: (1) small error probability if the algorithm is stopped prematurely, and (2) convergence to the best action if enough time is given.

Besides MPDs, we are also interested in game-tree search. Over the years, Monte-Carlo simulation based search algorithms have been used successfully in many non-deterministic and imperfect information games, including backgammon [14], poker [4] and Scrabble [12]. Recently, Monte-Carlo search proved to be competitive in deterministic games with large branching factors, viz. in Go [5]. For real-time strategy games, due to their enormous branching factors and stochasticity, Monte-Carlo simulations seems to be one of the few feasible approaches for planning [7]. Intriguingly, Monte-Carlo search algorithms used by today's games programs use either uniform sampling of actions or some heuristic biasing of the action selection probabilities that come with no guarantees.

The main idea of the algorithm proposed in this paper is to sample actions selectively. In order to motivate our approach let us consider problems with a large number of actions and assume that the lookahead is carried out at a fixed depth D. If sampling can be restricted to say half of the actions at all stages then the overall work reduction is $(1/2)^D$. Hence, if one is able to identify a large subset of the suboptimal actions early in the sampling procedure then huge performance improvements can be expected.

By definition, an action is suboptimal for a given state, if its value is less than the best of the action-values for the same state. Since action-values depend on the values of successor states, the problem boils down to getting the estimation error of the state-values for such states decay fast. In order to achieve this, an efficient algorithm must balance between testing alternatives that look currently the best so as to obtain precise estimates, and the exploration of currently suboptimal-looking alternatives, so as to ensure that no good alternatives are missed because of early estimation errors. Obviously, these criteria are contradictory and the problem of finding the right balance is known as the the exploration-exploitation dilemma. The most basic form of this dilemma shows up in *multi-armed bandit problems* [1].

The main idea in this paper it to apply a particular bandit algorithm, UCB1 (UCB stands for Upper Confidence Bounds), for rollout-based Monte-Carlo planning. The new algorithm, called UCT (UCB applied to trees) described in Section 2 is called UCT. Theoretical results show that the new algorithm is consistent, whilst experimental results (Section 3) for artificial game domains (P-games) and the sailing domain (a specific MDP) studied earlier in a similar context by others [11] indicate that UCT has a significant performance advantage over its closest competitors.

[1] In fact, as also noted by [8] the bound might be unimprovable, though this still remains an open problem.

2 The UCT algorithm

2.1 Rollout-based planning

In this paper we consider Monte-Carlo planning algorithms that we call *rollout-based*. As opposed to the algorithm described in the introduction (stage-wise tree building), a rollout-based algorithm builds its lookahead tree by repeatedly sampling episodes from the initial state. An episode is a sequence of state-action-reward triplets that are obtained using the domains generative model. The tree is built by adding the information gathered during an episode to it in an incremental manner.

The reason that we consider rollout-based algorithms is that they allow us to keep track of estimates of the actions' values at the sampled states encountered in earlier episodes. Hence, if some state is reencountered then the estimated action-values can be used to bias the choice of what action to follow, potentially speeding up the convergence of the value estimates. If the portion of states that are encountered multiple times in the procedure is small then the performance of rollout-based sampling degenerates to that of vanilla (non-selective) Monte-Carlo planning. On the other hand, for domains where the set of successor states concentrates to a few states only, rollout-based algorithms implementing selective sampling might have an advantage over other methods.

The generic scheme of rollout-based Monte-Carlo planning is given in Figure 1. The algorithm iteratively generates episodes (line 3), and returns the action with the highest average observed long-term reward (line 5).[2] In procedure UpdateValue the total reward q is used to adjust the estimated value for the given state-action pair at the given depth, completed by increasing the counter that stores the number of visits of the state-action pair at the given depth. Episodes are generated by the *search* function that selects and effectuates actions recursively until some terminal condition is satisfied. This can be the reach of a terminal state, or episodes can be cut at a certain depth (line 8). Alternatively, as suggested by Peret and Garcia [11] and motivated by iterative deepening, the search can be implemented in phases where in each phase the depth of search is inceased. An approximate way to implement iterative deepening, that we also follow in our experiments, is to stop the episodes with probability that is inversely proportional to the number of visits to the state.

The effectiveness of the whole algorithm will crucially depend on how the actions are selected in line 9. In vanilla Monte-Carlo planning (referred by MC in the following) the actions are sampled uniformly. The main contribution of the present paper is the introduction of a bandit-algorithm for the implementation of the selective sampling of actions.

2.2 Stochastic bandit problems and UCB1

A bandit problem with K arms (actions) is defined by the sequence of random payoffs X_{it}, $i = 1, \ldots, K$, $t \geq 1$, where each i is the index of a gambling machine

[2] The function *bestMove* is trivial, and is omitted due to the lack of space.

```
1: function MonteCarloPlanning(state)
2: repeat
3:     search(state, 0)
4: until Timeout
5: return bestAction(state,0)

6: function search(state, depth)
7: if Terminal(state) then return 0
8: if Leaf(state, d) then return Evaluate(state)
9: action := selectAction(state, depth)
10: (nextstate, reward) := simulateAction(state, action)
11: q := reward + γ search(nextstate, depth + 1)
12: UpdateValue(state, action, q, depth)
13: return q
```

Fig. 1. The pseudocode of a generic Monte-Carlo planning algorithm.

(the "arm" of a bandit). Successive plays of machine i yield the payoffs X_{i1}, X_{i2}, ... For simplicity, we shall assume that X_{it} lies in the interval $[0, 1]$. An allocation policy is a mapping that selects the next arm to be played based on the sequence of past selections and payoffs obtained. The expected regret of an allocation policy A after n plays is defined by $R_n = \max_i \mathbb{E}\left[\sum_{t=1}^{n} X_{it}\right] - \mathbb{E}\left[\sum_{j=1}^{K} \sum_{t=1}^{T_j(n)} X_{j,t}\right]$, where $I_t \in \{1, \ldots, K\}$ is the index of the arm selected at time t by policy A, and where $T_i(t) = \sum_{s=1}^{t} \mathbb{I}(I_s = i)$ is the number of times arm i was played up to time t (including t). Thus, the regret is the loss caused by the policy not always playing the best machine. For a large class of payoff distributions, there is no policy whose regret would grow slower than $O(\ln n)$ [10]. For such payoff distributions, a policy is said to resolve the exploration-exploitation tradeoff if its regret growth rate is within a constant factor of the best possible regret rate.

Algorithm UCB1, whose finite-time regret is studied in details by [1] is a simple, yet attractive algorithm that succeeds in resolving the exploration-exploitation tradeoff. It keeps track the average rewards $\overline{X}_{i,T_i(t-1)}$ for all the arms and chooses the arm with the best upper confidence bound:

$$I_t = \underset{i \in \{1,\ldots,K\}}{\operatorname{argmax}} \left\{ \overline{X}_{i,T_i(t-1)} + c_{t-1,T_i(t-1)} \right\}, \tag{1}$$

where $c_{t,s}$ is a bias sequence chosen to be

$$c_{t,s} = \sqrt{\frac{2 \ln t}{s}}. \tag{2}$$

The bias sequence is such that if X_{it} were independantly and identically distributed then the inequalities

$$\mathbb{P}\left(\overline{X}_{is} \geq \mu_i + c_{t,s}\right) \leq t^{-4}, \tag{3}$$

$$\mathbb{P}\left(\overline{X}_{is} \leq \mu_i - c_{t,s}\right) \leq t^{-4} \tag{4}$$

were satisfied. This follows from Hoeffding's inequality. In our case, UCB1 is used in the internal nodes to select the actions to be sampled next. Since for

any given node, the sampling probability of the actions at nodes below the node (in the tree) is changing, the payoff sequences experienced will drift in time. Hence, in UCT, the above expression for the bias terms $c_{t,s}$ needs to replaced by a term that takes into account this drift of payoffs. One of our main results will show despite this drift, bias terms of the form $c_{t,s} = 2C_p\sqrt{\frac{\ln t}{s}}$ with appropriate constants C_p can still be constructed for the payoff sequences experienced at any of the internal nodes such that the above tail inequalities are still satisfied.

2.3 The proposed algorithm

In UCT the action selection problem as treated as a separate multi-armed bandit for every (explored) internal node. The arms correspond to actions and the payoffs to the cumulated (discounted) rewards of the paths originating at the node.s In particular, in state s, at depth d, the action that maximizes $Q_t(s,a,d)+c_{N_{s,d}(t),N_{s,a,d}(t)}$ is selected, where $Q_t(s,a,d)$ is the estimated value of action a in state s at depth d and time t, $N_{s,d}(t)$ is the number of times state s has been visited up to time t at depth d and $N_{s,a,d}(t)$ is the number of times action a was selected when state s has been visited, up to time t at depth d.[3]

2.4 Theoretical analysis

The analysis is broken down to first analysing UCB1 for non-stationary bandit problems where the payoff sequences might drift, and then showing that the payoff sequences experienced at the internal nodes of the tree satisfy the drift-conditions (see below) and finally proving the consistency of the whole procedure.

The so-called drift-conditions that we make on the non-stationary payoff sequences are as follows: For simplicity, we assume that $0 \le X_{it} \le 1$. We assume that the expected values of the averages $\overline{X}_{in} = \frac{1}{n}\sum_{t=1}^{n} X_{it}$ converge. We let $\mu_{in} = \mathbb{E}\left[\overline{X}_{in}\right]$ and $\mu_i = \lim_{n\to\infty}\mu_{in}$. Further, we define δ_{in} by $\mu_{in} = \mu_i + \delta_{in}$and assume that the tail inequalities (3),(4) are satisfied for $c_{t,s} = 2C_p\sqrt{\frac{\ln t}{s}}$ with an appropriate constant $C_p > 0$. Throughout the analysis of non-stationary bandit problems we shall always assume without explicitly stating it that these drift-conditions are satisfied for the payoff sequences.

For the sake of simplicity we assume that there exist a single optimal action.[4] Quantities related to this optimal arm shall be upper indexed by a star, e.g., μ^*, $T^*(t)$, $\overline{X}^*_t$, etc. Due to the lack of space the proofs of most of the results are omitted.

We let $\Delta_i = \mu^* - \mu_i$. We assume that C_p is such that there exist an integer N_0 such that for $s \ge N_0$, $c_{ss} \ge 2|\delta_{is}|$ for any suboptimal arm i. Clearly, when UCT is applied in a tree then at the leafs $\delta_{is} = 0$ and this condition is automatically satisfied with $N_0 = 1$. For upper levels, we will argue by induction by showing an upper bound δ_{ts} for the lower levels that C_p can be selected to make $N_0 < +\infty$.

Our first result is a generalization of Theorem 1 due to Auer et al. [1]. The proof closely follows this earlier proof.

[3] The algorithm has to be implemented such that division by zero is avoided.

[4] The generalization of the results to the case of multiple optimal arms follow easily.

Theorem 1 *Consider UCB1 applied to a non-stationary problem. Let $T_i(n)$ denote the number of plays of arm i. Then if i the index of a suboptimal arm, $n > K$, then $\mathbb{E}[T_i(n)] \leq \frac{16C_p^2 \ln n}{(\Delta_i/2)^2} + 2N_0 + \frac{\pi^2}{3}$.*

At those internal nodes of the lookahead tree that are labelled by some state, the state values are estimated by averaging all the (cumulative) payoffs for the episodes starting from that node. These values are then used in calculating the value of the action leading to the given state. Hence, the rate of convergence of the bias of the estimated state-values will influence the rate of convergence of values further up in the tree. The next result, building on Theorem 1, gives a bound on this bias.

Theorem 2 *Let $\overline{X}_n = \sum_{i=1}^{K} \frac{T_i(n)}{n} \overline{X}_{i,T_i(n)}$. Then*

$$\left|\mathbb{E}\left[\overline{X}_n\right] - \mu^*\right| \leq |\delta_n^*| + O\left(\frac{K(C_p^2 \ln n + N_0)}{n}\right), \tag{5}$$

UCB1 never stops exploring. This allows us to derive that the average rewards at internal nodes concentrate quickly around their means. The following theorem shows that the number of times an arm is pulled can actually be lower bounded by the logarithm of the total number of trials:

Theorem 3 (Lower Bound) *There exists some positive constant ρ such that for all arms i and n, $T_i(n) \geq \lceil \rho \log(n) \rceil$.*

Among the the drift-conditions that we made on the payoff process was that the average payoffs concentrate around their mean quickly. The following result shows that this property is kept intact and in particular, this result completes the proof that if payoff processes futher down in the tree satisfy the drift conditions then payoff processes higher in the tree will satisfy the drift conditions, too:

Theorem 4 *Fix $\delta > 0$ and let $\Delta_n = 9\sqrt{2n \ln(2/\delta)}$. The following bounds hold true provided that n is sufficiently large: $\mathbb{P}\left(n\overline{X}_n \geq n\mathbb{E}\left[\overline{X}_n\right] + \Delta_n\right) \leq \delta$, $\mathbb{P}\left(n\overline{X}_n \leq n\mathbb{E}\left[\overline{X}_n\right] - \Delta_n\right) \leq \delta$.*

Finally, as we will be interested in the failure probability of the algorithm at the root, we prove the following result:

Theorem 5 (Convergence of Failure Probability) *Let $\hat{I}_t = \mathrm{argmax}_i \, \overline{X}_{i,T_i(t)}$. Then $P(\hat{I}_t \neq i^*) \leq C\left(\frac{1}{t}\right)^{\frac{\rho}{2}\left(\frac{\min_{i \neq i^*} \Delta_i}{36}\right)^2}$ with some constant C. In particular, it holds that $\lim_{t\to\infty} P(\hat{I}_t \neq i^*) = 0$.*

Now follows our main result:

Theorem 6 *Consider a finite-horizon MDP with rewards scaled to lie in the $[0,1]$ interval. Let the horizon of the MDP be D, and the number of actions per state be K. Consider algorithm UCT such that the bias terms of UCB1 are multiplied by D. Then the bias of the estimated expected payoff, $\overline{X}_n$, is $O(\log(n)/n)$. Further, the failure probability at the root converges to zero at a polynomial rate as the number of episodes grows to infinity.*

Proof. (Sketch) The proof is done by induction on D. For $D = 1$ UCT just corresponds to UCB1. Since the tail conditions are satisfied with $C_p = 1/\sqrt{2}$ by Hoeffding's inequality, the result follows from Theorems 2 and 5.

Now, assume that the result holds for all trees of up to depth $D - 1$ and consider a tree of depth D. First, divide all rewards by D, hence all the cumulative rewards are kept in the interval $[0, 1]$. Consider the root node. The result follows by Theorems 2 and 5 provided that we show that UCT generates a non-stationary payoff sequence at the root satisfying the drift-conditions. Since by our induction hypothesis this holds for all nodes at distance one from the root, the proof is finished by observing that Theorem 2 and 4 do indeed ensure that the drift conditions are satisfied. The particular rate of convergence of the bias is obtained by some straightforward algebra.

By a simple argument, this result can be extended to discounted MDPs. Instead of giving the formal result, we note that if some desired accuracy, ϵ_0 is fixed, similarly to [8] we may cut the search at the effective ϵ_0-horizon to derive the convergence of the action values at the initial state to the ϵ_0-vicinity of their true values. Then, similarly to [8], given some $\epsilon > 0$, by choosing ϵ_0 small enough, we may actually let the procedure select an ϵ-optimal action by sampling a sufficiently large number of episodes (the actual bound is similar to that obtained in [8]).

3 Experiments

3.1 Experiments with random game trees

A P-game tree [13] is a minimax tree that is meant to model games where at the end of the game the winner is decided by a global evaluation of the board position where some counting method is employed (examples of such games include Go, Amazons and Clobber). Accordingly, rewards are only associated with transitions to terminal states. These rewards are computed by first assigning values to moves (the moves are deterministic) and summing up the values along the path to the terminal state.[5] If the sum is positive, the result is a win for MAX, if it is negative the result is a win for MIN, whilst it is draw if the sum is 0. In the experiments, for the moves of MAX the move value was chosen uniformly from the interval $[0, 127]$ and for MIN from the interval $[-127, 0]$.[6] We have performed experiments for measuring the convergence rate of the algorithm.[7]

First, we compared the performance of four search algorithms: alpha-beta (AB), plain Monte-Carlo planning (MC), Monte-Carlo planning with minimax value update (MMMC), and the UCT algorithm. The failure rates of the four algorithms are plotted as function of iterations in Figure 2. Figure 2, left corresponds to trees with branching factor (B) two and depth (D) twenty, and

[5] Note that the move values are not available to the player during the game.

[6] This is different from [13], where 1 and -1 was used only.

[7] Note that for P-games UCT is modified to a negamax-style: In MIN nodes the negative of estimated action-values is used in the action selection procedures.

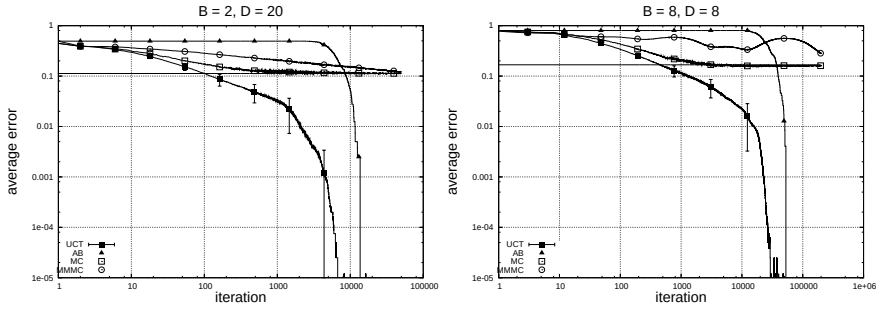

Fig. 2. Failure rate in P-games. The 95% confidence intervals are also shown for UCT.

Figure 2, right to trees with branching factor eight and depth eight. The failure rate represents the frequency of choosing the incorrect move if stopped after a number of iterations. For alpha-beta it is assumed that it would pick a move randomly, if the search has not been completed within a number of leaf nodes.[8] Each data point is averaged over 200 trees, and 200 runs for each tree. We observe that for both tree shapes UCT is converging to the correct move (i.e. zero failure rate) within a similar number of leaf nodes as alpha-beta does. Moreover, if we accept some small failure rate, UCT may even be faster. As expected, MC is converging to failure rate levels that are significant, and it is outperformed by UCT even for smaller searches. We remark that failure rate for MMCS is higher than for MC, although MMMC would eventually converge to the correct move if run for enough iterations.

Second, we measured the convergence rate of UCT as a function of search depth and branching factor. The required number of iterations to obtain failure rate smaller than some fixed value is plotted in Figure 3. We observe that for P-game trees UCT is converging to the correct move in order of $B^{D/2}$ number of iterations (the curve is roughly parallel to $B^{D/2}$ on log-log scale), similarly to alpha-beta. For higher failure rates, UCT seems to converge faster than $o(B^{D/2})$.

Note that, as remarked in the discussion at the end of Section 2.4, due to the faster convergence of values for deterministic problems, it is natural to decay the bias sequence with distance from the root (depth). Accordingly, in the experiments presented in this section the bias $c_{t,s}$ used in UCB was modified to $c_{t,s} = (\ln t/s)^{(D+d)/(2D+d)}$, where D is the estimated game length starting from the node, and d is the depth of the node in the tree.

3.2 Experiments with MDPs

The sailing domain [11, 15] is a finite state- and action-space stochastic shortest path problem (SSP), where a sailboat has to find the shortest path between two

[8] We also tested choosing the best looking action based on the incomplete searches. It turns out, not unexpectedly, that this choice does not influence the results.

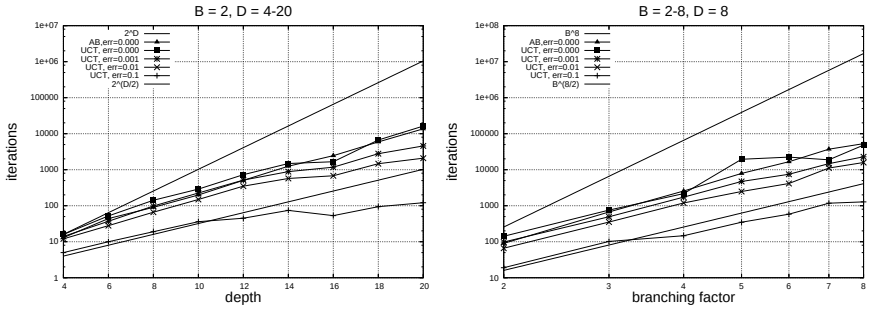

Fig. 3. P-game experiment: Number of episodes as a function of failure probability.

points of a grid under fluctuating wind conditions. This domain is particularly interesting as at present SSPs lie outside of the scope of our theoretical results.

The details of the problem are as follows: the sailboat's position is represented as a pair of coordinates on a grid of finite size. The controller has 7 actions available in each state, giving the direction to a neighbouring grid position. Each action has a cost in the range of 1 and 8.6, depending on the direction of the action and the wind: The action whose direction is just the opposite of the direction of the wind is forbidden. Following [11], in order to avoid issues related to the choice of the evaluation function, we construct an evaluation function by randomly perturbing the optimal evaluation function that is computed off-line by value-iteration. The form of the perturbation is $\hat{V}(x) = (1 + \epsilon(x))V^*(x)$, where x is a state, $\epsilon(x)$ is a uniform random variable drawn in $[-0.1; 0.1]$ and $V^*(x)$ is the optimal value function. The assignment of specific evaluation values to states is fixed for a particular run. The performance of a stochastic policy is evaluated by the error term $Q^*(s, a) - V^*(s)$, where a is the action suggested by the policy in state s and Q^* gives the optimal value of action a in state s. The error is averaged over a set of 1000 randomly chosen states.

Three planning algorithms are tested: UCT, ARTDP [3], and PG-ID [11]. For UCT, the algorithm described in Section 2.1 is used. The episodes are stopped with probability $1/N_s(t)$, and the bias is multiplied (heuristically) by 10 (this multiplier should be an upper bound on the total reward).[9] For ARTDP the evaluation function is used for initializing the state-values. Since these values are expected to be closer to the true optimal values, this can be expected to speed up convergence. This was indeed observed in our experiments (not shown here). Moreover, we found that Boltzmann-exploration gives the best performance with ARTDP and thus it is used in this experiment (the 'temperature' parameter is kept at a fixed value, tuned on small-size problems). For PG-ID the same parameter setting is used as [11].

[9] We have experimented with alternative stopping schemes. No major differences were found in the performance of the algorithm for the different schemes. Hence these results are not presented here.

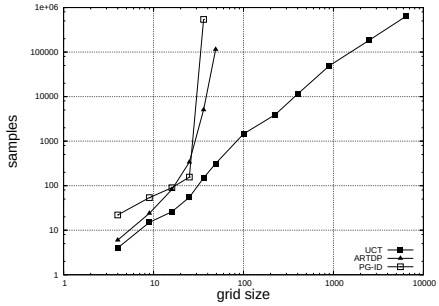

Fig. 4. Number of samples needed to achieve an error of size 0.1 in the sailing domain. 'Problem size' means the size of the grid underlying the state-space. The size of the state-space is thus 24×'problem size', since the wind may blow from 8 directions, and 3 additional values (per state) give the 'tack'.

Since, the investigated algorithms are building rather non-uniform search trees, we compare them by the total number of samples used (this is equal to the number of calls to the simulator). The required number of samples to obtain error smaller than 0.1 for grid sizes varying from 2×2 to 40×40 is plotted in Figure 4. We observe that UCT requires significantly less samples to achieve the same error than ARTDP and PG-ID. At least on this domain, we conclude that UCT scales better with the problem size than the other algorithms.

3.3 Related research

Besides the research already mentioned on Monte-Carlo search in games and the work of [8], we believe that the closest to our work is the work of [11] who also proposed to use rollout-based Monte-Carlo planning in undiscounted MDPs with selective action sampling. They compared three strategies: uniform sampling (uncontrolled search), Boltzmann-exploration based search (the values of actions are transformed into a probability distribution, i.e., samples better looking actions are sampled more often) and a heuristic, interval-estimation based approach. They observed that in the 'sailing' domain lookahead pathologies are present when the search is uncontrolled. Experimentally, both the interval-estimation and the Boltzmann-exploration based strategies were shown to avoid the lookahead pathology and to improve upon the basic procedure by a large margin. We note that Boltzmann-exploration is another widely used bandit strategy, known under the name of "exponentially weighted average forecaster" in the on-line prediction literature (e.g. [2]). Boltzmann-exploration as a bandit strategy is inferior to UCB in stochastic environments (its regret grows with the square root of the number of samples), but is preferable in adversary environments where UCB does not have regret guarantees. We have also experimented with a Boltzmann-exploration based strategy and found that in the case of our domains it performs significantly weaker than the upper-confidence value based algorithm described here.

Recently, Chang et al. also considered the problem of selective sampling in finite horizon undiscounted MDPs [6]. However, since they considered domains where there is little hope that the same states will be encountered multiple times, their algorithm samples the tree in a depth-first, recursive manner: At each node they sample (recursively) a sufficient number of samples to compute a good approximation of the value of the node. The subroutine returns with an approximate evaluation of the value of the node, but the returned values are not stored (so when a node is revisited, no information is present about which actions can be expected to perform better). Similar to our proposal, they suggest to propagate the average values upwards in the tree and sampling is controlled by upper-confidence bounds. They prove results similar to ours, though, due to the independence of samples the analysis of their algorithm is significantly easier. They also experimented with propagating the maximum of the values of the children and a number of combinations. These combinations outperformed propagating the maximum value. When states are not likely to be encountered multiple times, our algorithm degrades to this algorithm. On the other hand, when a significant portion of states (close to the initial state) can be expected to be encountered multiple times then we can expect our algorithm to perform significantly better.

4 Conclusions

In this article we introduced a new Monte-Carlo planning algorithm, UCT, that applies the bandit algorithm UCB1 for guiding selective sampling of actions in rollout-based planning. Theoretical results were presented showing that the new algorithm is consistent in the sense that the probability of selecting the optimal action can be made to converge to 1 as the number of samples grows to infinity.

The performance of UCT was tested experimentally in two synthetic domains, viz. random (P-game) trees and in a stochastic shortest path problem (sailing). In the P-game experiments we have found that the empirically that the convergence rates of UCT is of order $B^{D/2}$, same as for alpha-beta search for the trees investigated. In the sailing domain we observed that UCT requires significantly less samples to achieve the same error level than ARTDP or PG-ID, which in turn allowed UCT to solve much larger problems than what was possible with the other two algorithms.

Future theoretical work should include analysing UCT in stochastic shortest path problems and taking into account the effect of randomized terminating condition in the analysis. We also plan to use UCT as the core search procedure of some real-world game programs.

5 Acknowledgements

We would like to acknowledge support for this project from the Hungarian National Science Foundation (OTKA), Grant No. T047193 (Cs. Szepesvári) and from the Hungarian Academy of Sciences (Cs. Szepesvári, Bolyai Fellowship).

References

1. P. Auer, N. Cesa-Bianchi, and P. Fischer. Finite time analysis of the multiarmed bandit problem. *Machine Learning*, 47(2-3):235–256, 2002.
2. P. Auer, N. Cesa-Bianchi, Y. Freund, and R.E. Schapire. The nonstochastic multiarmed bandit problem. *SIAM Journal on Computing*, 32:48–77, 2002.
3. A.G. Barto, S.J. Bradtke, and S.P. Singh. Real-time learning and control using asynchronous dynamic programming. Technical report 91-57, Computer Science Department, University of Massachusetts, 1991.
4. D. Billings, A. Davidson, J. Schaeffer, and D. Szafron. The challenge of poker. *Artificial Intelligence*, 134:201–240, 2002.
5. B. Bouzy and B. Helmstetter. Monte Carlo Go developments. In H.J. van den Herik, H. Iida, and E.A. Heinz, editors, *Advances in Computer Games 10*, pages 159–174, 2004.
6. H.S. Chang, M. Fu, J. Hu, and S.I. Marcus. An adaptive sampling algorithm for solving Markov decision processes. *Operations Research*, 53(1):126–139, 2005.
7. M. Chung, M. Buro, and J. Schaeffer. Monte Carlo planning in RTS games. In *CIG 2005, Colchester, UK*, 2005.
8. M. Kearns, Y. Mansour, and A.Y. Ng. A sparse sampling algorithm for near-optimal planning in large Markovian decision processes. In *Proceedings of IJCAI'99*, pages 1324–1331, 1999.
9. L. Kocsis, Cs. Szepesvári, and J. Willemson. UCT: Bandit based Monte-Carlo planning in games. 2006. (in preparation), available at http://zaphod.aml.sztaki.hu/wg/index.pl/kocsis.
10. T.L. Lai and H. Robbins. Asymptotically efficient adaptive allocation rules. *Advances in Applied Mathematics*, 6:4–22, 1985.
11. L. Péret and F. Garcia. On-line search for solving Markov decision processes via heuristic sampling. In R.L. de Mántaras and L. Saitta, editors, *ECAI*, pages 530–534, 2004.
12. B. Sheppard. World-championship-caliber Scrabble. *Artificial Intelligence*, 134(1–2):241–275, 2002.
13. S.J.J. Smith and D.S. Nau. An analysis of forward pruning. In *AAAI*, pages 1386–1391, 1994.
14. G. Tesauro and G.R. Galperin. On-line policy improvement using Monte-Carlo search. In M.C. Mozer, M.I. Jordan, and T. Petsche, editors, *NIPS 9*, pages 1068–1074, 1997.
15. R. Vanderbei. Optimal sailing strategies, statistics and operations research program, 1996. University of Princeton, http://www.sor.princeton.edu/~rvdb/sail/sail.html.