Continuous Upper Confidence Trees

Adrien Coutoux, Jean-Baptiste Hoock, Nataliya Sokolovska, Olivier Teytaud, Nicolas Bonnard

Continuous Upper Confidence Trees

Adrien Couëtoux[1,2], Jean-Baptiste Hoock[1], Nataliya Sokolovska[1], Olivier Teytaud[1], and Nicolas Bonnard[2]

[1] TAO-INRIA, LRI, CNRS UMR 8623,
Université Paris-Sud, Orsay, France
[2] Artelys, 12 rue du Quatre Septembre Paris, France

Abstract. Upper Confidence Trees are a very efficient tool for solving Markov Decision Processes; originating in difficult games like the game of Go, it is in particular surprisingly efficient in high dimensional problems. It is known that it can be adapted to continuous domains in some cases (in particular continuous action spaces). We here present an extension of Upper Confidence Trees to continuous stochastic problems. We (i) show a deceptive problem on which the classical Upper Confidence Tree approach does not work, even with arbitrarily large computational power and with progressive widening (ii) propose an improvement, termed double-progressive widening, which takes care of the compromise between variance (we want infinitely many simulations for each action/state) and bias (we want sufficiently many nodes to avoid a bias by the first nodes) and which extends the classical progressive widening (iii) discuss its consistency and show experimentally that it performs well on the deceptive problem and on experimental benchmarks. We guess that the double-progressive widening trick can be used for other algorithms as well, as a general tool for ensuring a good bias/variance compromise in search algorithms.

1 Introduction

Monte-Carlo Tree Search [3] is now widely accepted as a great tool for high-dimensional games [9] and high-dimensional planning [10]; its most well known variant is Upper Confidence Trees [7]. It is already adapted to continuous domains [12, 11], but not for arbitrary stochastic transitions; this paper is devoted to this extension.

In section 2, we will present Progressive Widening (PW), a classical improvement of UCT in continuous or large domains. We will see that PW is not sufficient for ensuring a good behavior in the most general setting; a simple but not trivial modification, termed double-PW, is proposed and validated. Experiments (section 3) will show that this modification makes UCT for Markov Decision Processes compliant with high-dimensional continuous domains with arbitrary stochastic transition.

In all the paper, $\#E$ denotes the cardinal of a set E.

2 Progressive widening for Upper Confidence Trees

Progressive strategies have been proposed in [4, 2] for tackling problems with big action spaces; they have been theoretically analyzed in [13], and used for continuous spaces in [11, 12]. We will here (i) define a variant of progressive widening (section 2.1), (ii) show why it can't be directly applied in some cases (section 2.2), (iii) define our version (section 2.3).

2.1 Progressive widening

Consider an algorithm, choosing between options $O = \{o_1, o_2, \dots, o_n, \dots\}$ at several time steps. More formally, this is as follows:

$R_0 = 0$
for $t = 1, t = 2, t = 3, \dots$ **do**
  Choose an option $o_{(t)} \in O$.
  Test it: get a reward r_t.
  Cumulate the reward: $R_t = R_{t-1} + r_t$.
end for

The goal is to design the "Choose" method so that the cumulated reward increases as fast as possible. An option (terminology of bandits) is equivalent to an action (terminology of reinforcement learning) or a move (terminology of games).

Many papers have been published on such problems, in particular around upper confidence bounds [8, 1]. Upper Confidence Bounds, in its simplest version, proceeds as follows:

Upper confidence bound algorithm.
$R_0 = 0$
for $t = 1, t = 2, t = 3, \dots$ **do**
  Choose an option $o_{(t)} \in O$ maximizing $score_t(o)$ defined as follows:
    $totalReward_t(o) = \sum_{1 \leq l \leq t-1, o_l = o} r_l$
    $nb_t(o) = \sum_{1 \leq l \leq t-1, o_l = o} 1$
    $score_t(o) = \frac{totalReward_t(o)}{nb_t(o)+1} + k\sqrt{\log(t)/(nb_t(o)+1)}$ $(+\infty$ if $nb_t(o) = 0)$
  Test it: get a reward r_t.
  Cumulate the reward: $R_t = R_{t-1} + r_t$.
end for

Variants of the score function are termed "bandit algorithms"; there are plenty of variants of the score formula; this is essentially independent of the aspects investigated in this paper.

A trouble in many mathematical works around such problems is that the set O is usually assumed small in front of the number of iterations. More precisely, the behavior of the algorithm above is trivial for $t \leq \#O$. [14] proposed the use of a constant s such that $nb_t(o) = 0 \Rightarrow score_t(o) = s$; this is the so-called First

Play Urgency algorithm. There are other specialized efficient tools for bandits used in "trees" such as rapid action value estimates [6, 5]; however these tools assume some sort of homogenity between the actions at various time steps. [3, 13, 2] proposed progressive strategies for big/infinite sets of arms. The principle is as follows for some constants $C > 0$ and $\alpha \in]0, 1[$ (as it is independent of the algorithm used for choosing an option, within a given pool of possible options, we do not explicitly write a score function as above):

Progressive widening with constants $C > 0$ **and** $\alpha \in]0, 1[$**.**
$R_0 = 0$
for $t = 1, t = 2, t = 3, \dots$ **do**
  Let $k = \lceil Ct^\alpha \rceil$.
  Choose an option $o_{(t)} \in \{o_1, \dots, o_k\}$.
  Test it: get a reward r_t.
  Cumulate the reward: $R_t = R_{t-1} + r_t$.
end for

The key point is that the chosen option is restricted to have index $\leq k$; the complete set $O = \{o_1, o_2, \dots\}$ is not allowed. This algorithm has the advantage that it is anytime: we do not have to know in advance at which value of t the algorithm will be stopped. [3] applied it successfully in the very efficient CrazyStone implementation of Monte-Carlo Tree Search [4]. Upper Confidence Tree (or Monte-Carlo Tree Search) is not a simple setting as above: when applying an option, we reach a new state; one can think of Monte-Carlo Tree Search (or UCT) as having one bandit in each possible state s of the reinforcement learning problem, for choosing between (infinitely many) options $o_1(s), o_2(s), \dots, o_n(s), \dots$. The algorithm is as follows, for a task in which all the reward is obtained in the final state[1]. The last line of the algorithm (returning the most simulated action from S) is often surprising for people who are not used to MCTS; it is known as much better than choosing the action with best expected reward.

[1] This assumption (that the reward is null except in the final state) simplifies the writing, but is not necessary for the work presented here.

Progressive Widening (PW) applied in state s with constants $C > 0$ and $\alpha \in]0, 1[$.
Input: a state s.
Output: an action.
Let $nbVisits(s) \leftarrow nbVisits(s) + 1$
and let $t = nbVisits(s)$
Let $k = \lceil Ct^{\alpha} \rceil$.
Choose an option $o_{(t)}(s) \in \{o_1(s), \ldots, o_k(s)\}$ maximizing $score_t(s, o)$ defined as follows:
$totalReward_t(s, o) = \sum_{1 \leq l \leq t-1, o_l(s)=o} r_l(s)$
$nb_t(s, o) = \sum_{1 \leq l \leq t-1, o_l=o} 1$
$score_t(s, o) = \frac{totalReward_t(s,o)}{nb_t(s,o)+1} + k\sqrt{\log(t)/(nb_t(s, o) + 1)}$
$(+\infty$ if $nb_t(o) = 0)$
Test it: get a state s'.

UCT algorithm with progressive widening
Input: a state S, a time budget.
Output: an action a.
Initialize: $\forall s, nbSims(s) = 0$
while Time not elapsed **do**
  // starting a simulation.
  $s = S$.
  while s is not a terminal state **do**
    Apply progressive widening in state s for choosing an option o.
    Let s' be the state reached from s when choosing action o.
    $s = s'$
  end while
  // the simulation is over; it started at S and reached a final state.
  Get a reward $r = Reward(s)$ // s is a final state, it has a reward.
  For all states s in the simulation above, let $r_{nbVisits(s)}(s) = r$.
end while
Return the action which was simulated most often from S.

It is important to keep in mind that the progressive widening algorithm is applied in each visited state; some states might be visited only once, or never, and some other states are visited very often. MCTS with progressive widening or progressive strategies is the only version of MCTS which works in continuous action spaces [12, 11]; however, it was applied only with the property that applying a given action a in a given state s can lead to finitely many states only. We will see in section 2.2 that this methodology (the algorithm above) does not work as is in the case in which there is a null probability of reaching twice the same state when applying the same action in the same state (i.e. typically it does not work for stochastic transitions with continuous support).

2.2 Why it does not work as is for randomized transitions in continuous domains

We have presented UCT with progressive widening. In this section we will show why it is not sufficient for a consistent behavior (i.e. for a convergence toward maximum expected reward) in some cases, in particular when the transitions are stochastic and never lead twice to the state - one can think of the case of a Gaussian additive noise, or any other noise such that states can be reached only once.

Let us assume now that we have an infinite (discrete or continuous) domain of options. This is not too much a trouble for progressive widening: if $\alpha \in]0,1[$, and if the o_t are a good approximation of the set of possible actions (typically, in continuous domains, we assume that the set $\{o_i; i \geq 1\}$ is dense in the set of actions and the reward has some smoothness properties), then asymptotically, good actions are explored, and all these explored actions are sampled infinitely often [13].

Let us now consider what happens if we have randomized transitions. Assume that for a state s and an action a, we can reach infinitely many transitions. Consider such a state s, and assume that we visit it infinitely often; we would like the algorithms to have two characteristics:

1. infinitely many actions $(a_1, \alpha_2, a_3, \dots)$ will be explored (for reducing the bias due to the choice of actions);
2. all states that can be reached from s are themselves explored infinitely often (for reducing the variance due to random exploration).

We do not have a mathematical proof that these two requirements are enough, but they look quite reasonnable: in order to approximate a continuous set of actions, and unless we have an efficient pruning to a finite set of actions, we will have to explore infinitely many actions; and if we consider only finitely many possible consequences of an action whereas the real support is infinite we will miss important facts and it is hard to believe that the algorithm can be consistent.

For classical score functions, progressive widening will ensure the first property. But the second property will not be ensured, as in continuous domains with stochastic transitions, nothing ensures that we will reach twice the same state, whenever we play infinitely often a given action a in a given state s. This will be illustrated on the Trap problem later.

The following section is devoted to proposing a solution to this problem.

2.3 Proposed solution: double progressive widening

Section 2.1 has presented the known form of progressive widening, and section 2.2 has shown that in some cases it does not work (namely, when there are pairs (s, a) such that, with probability one, applying a in s infinitely often does not lead to visiting the following states infinitely often). In this section, we

propose the use of a second form of progressive widening in MCTS, as follows:

Double Progressive Widening (DPW) applied in state s with constants $C > 0$ and $\alpha \in]0, 1[$.
Input: a state s.
Output: a state s'.
Let $nbVisits(s) \leftarrow nbVisits(s) + 1$
and let $t = nbVisits(s)$
Let $k = \lceil Ct^{\alpha} \rceil$.
Choose an option $o_{(t)}(s) \in \{o_1(s), \ldots, o_k(s)\}$ maximizing $score_t(s, o)$ defined as follows:
$totalReward_t(s, o) = \sum_{1 \leq l \leq t-1, o_l(s)=o} r_l(s)$
$nb_t(s, o) = \sum_{1 \leq l \leq t-1, o_{(l)}(s)=o} 1$
$score_t(s, o) = \frac{totalReward_t(s,o)}{nb_t(s,o)+1} + k\sqrt{\log(t)/(nb_t(s, o) + 1)}$ ($+\infty$ if $nb_t(o) = 0$)
Let $k' = \lceil Cnb_t(s, o_{(t)}(s))^{\alpha} \rceil$
if $k' > \#Children_t(s, o_{(t)}(s))$ // progressive widening on the random part **then**
  Test option $o_{(t)}(s)$; get a new state s'
  if $s' \notin Children_t(s, o_{(t)})$ **then**
    $Children_{t+1}(s, o_{(t)}) = Children_t(s, o_{(t)}) \cup \{s'\}$
  else
    $Children_{t+1}(s, o_{(t)}) = Children_t(s, o_{(t)})$
  end if
else
  $Children_{t+1}(s, o_{(t)}) = Children_t(s, o_{(t)})$
  Choose s' in $Children_t(s, o_{(t)})$ // s' is chosen with probability $nb_t(s, o, s')/nb_t(s, o)$
end if

UCT algorithm with DPW
Input: a state S.
Output: an action a.
Initialize: $\forall s, nbSims(s) = 0$
while Time not elapsed **do**
  // starting a simulation.
  $s = S$.
  while s is not a terminal state **do**
    Apply DPW in state s for choosing an option o.
    Let s' be the state given by DPW.
    $s = s'$
  end while
  // the simulation is over; it started at S and reached a final state.
  Get a reward $r = Reward(s)$ // s is a final state, it has a reward.
  For all states s in the simulation above, let $r_{nbVisits(s)}(s) = r$.
end while
Return the action which was simulated most often from S.

This algorithm is not so intuitive, for the second progressive widening part. The idea is as follows:

– If k' is large enough, we consider adding one more child to the pool of visited children: we simulate a transition and get a state s'. If we get an already visited child, then we go to this child; otherwise, we create a new child.
– If k' is not large enough, then we sample one of the previously seen children. As they are not necessarily equally likely, we select a child proportionally to the number of times it has been generated.

The algorithm has been designed with a "consistency" objective in mind, which is twofolds:

– Infinite visiting: we want that if a node is visited infinitely often, then we generate infinitely many children, and each of these children is itself visited infinitely often. By induction, this property ensures that all created nodes are visited infinitely often. Progressive widening and the UCB formula (or many other formulas in fact) ensure this property.
– Propagation: the average reward of any node visited infinitely often converges to a limit and this limit (for a non-terminal node) is the average reward corresponding to its children which have best asymptotic average reward. This property is ensured by the careful sampling in the progressive widening.

3 Experiments

In section 3.1 we present a deceptive problem designed specifically for pointing out the inconsistency of the classical PW approach. In section 3.2 we treat a more real problem.

3.1 Trap problem

In this section we present the toy problem, aimed at being (i) deceptive for the simple progressive widening (ii) as simple as possible. We provide our experimental results as well.

Problem description. This problem has been designed to clearly illustrate the weakness of the simple progressive widening. In this problem, one has to make two successive decisions, in order to maximize the reward. As we will see, the optimal policy is to make a risky move at the first step, in order to be able to obtain the maximum reward on the second (and last) step. The state will be denoted x, and is initialized at $x_0 = 0$. At each time step t the decision is denoted $d_t \in [0, 1]$. Let $R > 0$ be the noise amplitude at each time step. At a time step t, given the current state x_t and a decision d_t, we have:

$$x_{t+1} = x_t + d_t + R \times Y,$$

Y being a random variable following a uniform distribution on $[0, 1]$.

The trap problem relies on five positive real numbers: the high reward h, the average reward a, the initial ramp length l, and the trap width w. The high

reward will be given if and only if we cross the trap, otherwise we obtain 0. If we stay on the initial ramp, we get the average reward. We thus define the reward function $r(\cdot)$ as follows:

$$r(x) = \begin{cases} a \text{ if } & x < l \\ 0 \text{ if } & l < x < l + w \\ h \text{ if } & x > l + w \end{cases}$$

The objective is to maximize $r(x_0) + r(x_1)$, the cumulated reward.

The shape of the reward function is shown in Fig.1.

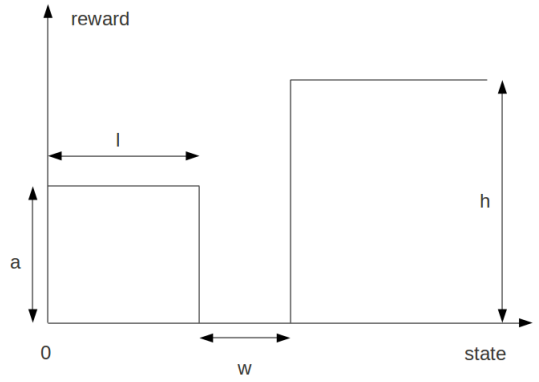

Fig. 1. Shape of the reward function: Trap problem.

Experimental results. We compare simple progressive widening and double progressive widening on the trap problem. In our experiments, we used the following settings: $a = 70$, $h = 100$, $l = 1$, $w = 0.7$, $R = 0.01$. With these parameters, the optimal behavior si to have the first decision $d_0 \in [0.7, 1]$ and $d_1 \geq 1.7 - d_0$. If one makes optimal decisions, one has an expected reward of $r^* = 170$. That is the reward toward which the Double progressive widening version of Monte Carlo Tree Search converges. However, the Simple progressive widening version does not reach this optimal reward. Worse, as we increase the computation time, it becomes less efficient, converging toward a local optimum, 140.

The mean values of the rewards are shown in Fig. 2 and the medians of the rewards are shown in Fig. 3. Each point is computed according to 100 simulations.

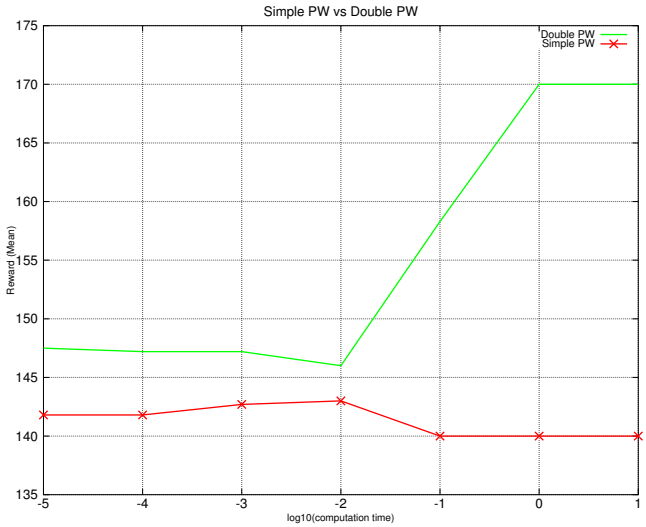

Fig. 2. Mean of the reward, for the trap problem with $a = 70$, $h = 100$, $l = 1$, $w = 0.7$, $R = 0.01$. The estimated standard deviations of the rewards are $STD_{DPW} = [13.06, 12.88, 12.88, 12.06, 14.70, 0, 0]$ for Double PW and $STD_{SPW} = [7.16, 7.16, 8.63, 9.05, 0, 0, 0]$ for Simple PW - the differences are clearly significant, where STD means standard deviation.

3.2 The power management problem

In this section we present a real world problem. We show our experimental results for various settings of the power management problem.

Problem description. The experimental setup is an energy stock management problem. We have finitely many energy stocks (nuclear stocks, water stocks), each of them can be used to produce electricity; we can also produce electricity with classical thermal plants, that are more costly. The problem is to find the right tradeoff between

- using stocks now (in order to save up money), with the risk that later we might have peaks of demands, leading to very high costs if we do not have enough stocks.
- keeping stocks for later (in order to avoid the trouble above), with the risk that we might have too much in a stock if there is no big peak of demand.

Also, even for a fixed amount of water used from the stocks, we have to decide which stock we want to use. In particular, stocks above a given level are lost (because we have to get rid of water when the level is too high). All stocks are

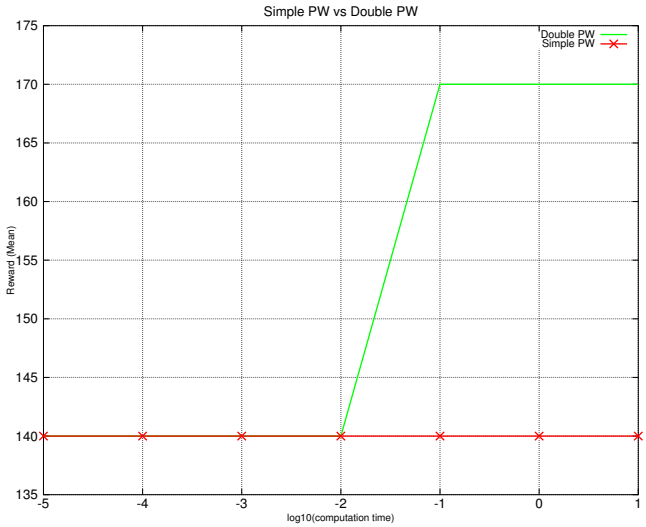

Fig. 3. Median of the reward, for the trap problem with $a = 70$, $h = 100$, $l = 1$, $w = 0.7$, $R = 0.01$

not equivalent: some of them have stronger inflows than others, and the used part of some stocks is transfered to other stocks whereas others are not or not to the same. One can think of a graph of reservoirs, water used in a given stock being forwarded to another stock given by the graph. In our implementation, the demand is a function of the time, determined in advance. The inflows, however, follow a lognormal distribution. Hence, they take different values from one simulation to the other.

The code of the problem can be found in `http://www.lri.fr/~couetoux/stock.cpp` or requested by email.

Small size experiments. We consider here 2 stocks only and 5 time steps. We compare the Q-learning algorithm from the Mash project `http://mash-project.eu/`, our progressive widening Monte-Carlo Tree Search approach, a greedy algorithm only maximizing the short term, and a blind planner optimizing a sequence of decisions regardless of stock levels. Results are presented in Fig. 4. We plot the median values of cumulated reward as a function of time. It is easy to see that the Simple and Double PW MCTS achieve the best performance, compared to Blind, Greedy, and Q-learning approaches. In this particular (power management) problem, decisions are strongly associated with stock levels (states). The performance of the Blind is poor, since it makes illegal decisions rather often. Our implementation of the Q-learning suffers from the same phenomenon.

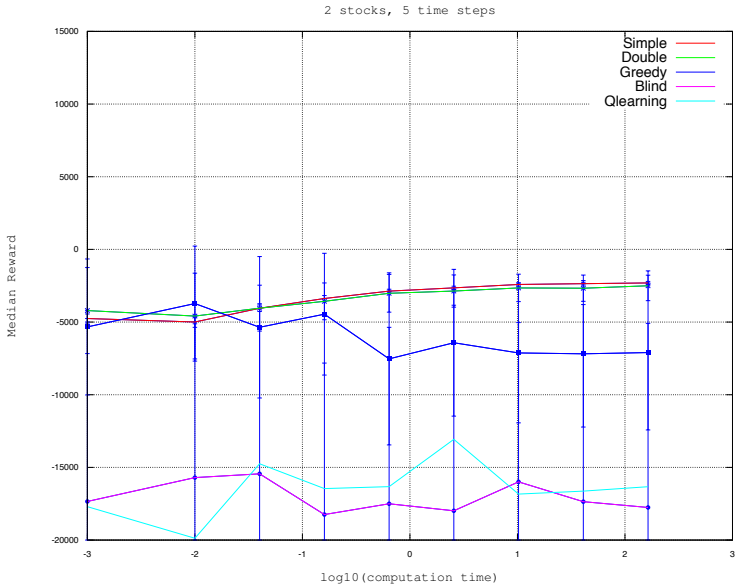

Fig. 4. The power management problem. Median values of cumulated reward with 2 stocks and 5 time steps.

Bigger size experiments. We here switch to 6 stocks and 21 time steps, corresponding to 3 time steps per day, one week, with an expected increase of demand at some point during the week. Results are presented in Fig. 5. Note, that the conclusion is the same as for the small scale problem, described above: the proposed Simple and Double PW MCTS are very competitive compared to other tested methods. We did not include the results of the Q-learning. In this setting, the Q-learning obtained rather poor rewards. For the sake of clarity of results of other approaches, we do not show the performance of the Q-learning on Fig 5.

4 Conclusion

We have modified progressive widening in order to make it compliant with continuous domains with general noise. Experimentally, the "double-PW" modification was very efficient on deceptive problems aimed at pointing out the weaknesses of simple PW; we conjecture that for some problems, both versions are roughly equivalent, and for some problems the double PW is much better. On the other hand, on a realistic problem, the modification had disappointingly little effect. The formal proof of the consistency of the double PW (i.e. the convergence to the optimal reward for wide classes of Markov Decision Processes) has not been given and is the main further work.

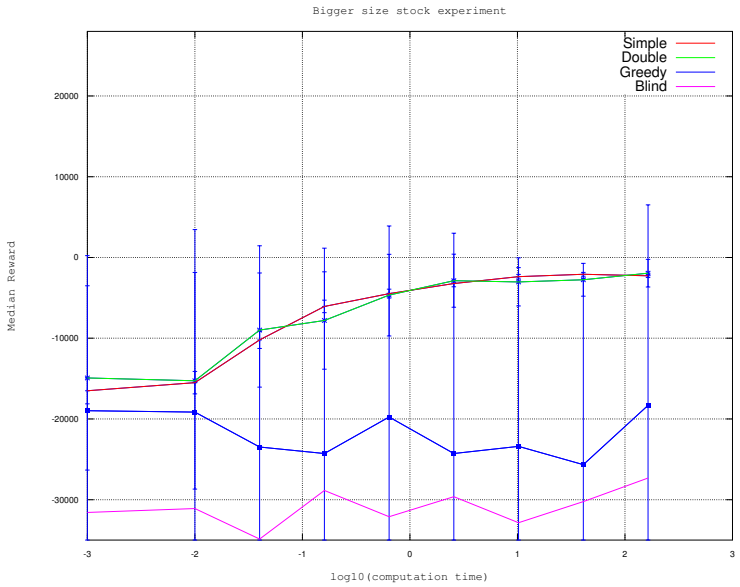

Fig. 5. The power management problem. Median values of cumulated reward. Experiments with 6 stocks and 21 time steps.

References

1. P. Auer. Using confidence bounds for exploitation-exploration trade-offs. *The Journal of Machine Learning Research*, 3:397–422, 2003.
2. G. Chaslot, M. Winands, J. Uiterwijk, H. van den Herik, and B. Bouzy. Progressive Strategies for Monte-Carlo Tree Search. In P. Wang et al., editors, *Proceedings of the 10th Joint Conference on Information Sciences (JCIS 2007)*, pages 655–661. World Scientific Publishing Co. Pte. Ltd., 2007.
3. R. Coulom. Efficient Selectivity and Backup Operators in Monte-Carlo Tree Search. *In P. Ciancarini and H. J. van den Herik, editors, Proceedings of the 5th International Conference on Computers and Games, Turin, Italy*, 2006.
4. R. Coulom. Computing elo ratings of move patterns in the game of go. In *Computer Games Workshop, Amsterdam, The Netherlands*, 2007.
5. H. Finnsson and Y. Björnsson. Simulation-based approach to general game playing. In *AAAI'08: Proceedings of the 23rd national conference on Artificial intelligence*, pages 259–264. AAAI Press, 2008.
6. S. Gelly and D. Silver. Combining online and offline knowledge in UCT. In *ICML '07: Proceedings of the 24th international conference on Machine learning*, pages 273–280, New York, NY, USA, 2007. ACM Press.
7. L. Kocsis and C. Szepesvari. Bandit based Monte-Carlo planning. In *15th European Conference on Machine Learning (ECML)*, pages 282–293, 2006.
8. T. Lai and H. Robbins. Asymptotically efficient adaptive allocation rules. *Advances in Applied Mathematics*, 6:4–22, 1985.

9. C.-S. Lee, M.-H. Wang, G. Chaslot, J.-B. Hoock, A. Rimmel, O. Teytaud, S.-R. Tsai, S.-C. Hsu, and T.-P. Hong. The Computational Intelligence of MoGo Revealed in Taiwan's Computer Go Tournaments. *IEEE Transactions on Computational Intelligence and AI in games*, 2009.
10. H. Nakhost and M. Müller. Monte-carlo exploration for deterministic planning. In C. Boutilier, editor, *IJCAI*, pages 1766–1771, 2009.
11. P. Rolet, M. Sebag, and O. Teytaud. Optimal active learning through billiards and upper confidence trees in continous domains. In *Proceedings of the ECML conference*, 2009.
12. P. Rolet, M. Sebag, and O. Teytaud. Optimal robust expensive optimization is tractable. In *Gecco 2009*, page 8 pages, Montréal Canada, 2009. ACM.
13. Y. Wang, J.-Y. Audibert, and R. Munos. Algorithms for infinitely many-armed bandits. In *Advances in Neural Information Processing Systems*, volume 21, 2008.
14. Y. Wang and S. Gelly. Modifications of UCT and sequence-like simulations for Monte-Carlo Go. In *IEEE Symposium on Computational Intelligence and Games, Honolulu, Hawaii*, pages 175–182, 2007.