Adding Double Progressive Widening to Upper Confidence Trees to Cope with Uncertainty in Planning Problems

Adrien Couëtoux[1,2] and Hassen Doghmen[1]

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

Abstract. Current state of the art methods in energy policy planning only approximate the problem (Linear Programming on a finite sample of scenarios, Dynamic Programming on an approximation of the problem, etc). Monte-Carlo Tree Search (MCTS [3]) seems to be a potential candidate to converge to an exact solution of these problems ([2]). But how fast, and how do key parameters (double/simple progressive widening) influence the rate of convergence (or even the convergence itself), are still open questions. Also, MCTS completely ignores the features of the problem, including the scale of the objective function. In this paper, we present MCTS, and its extension to continuous/stochastic domains. We show that on problems with continuous action spaces and infinite support of random variables, the "vanilla" version of MCTS fails. We also show how the double progressive widening technique success[2] relies on its widening coefficient. We also study the impact of an unknown variance of the random variables, to see if it affects the optimal choice of the widening coefficients.

Keywords: Stochastic Planning, Exploration/Exploitation

1 Introduction

Monte Carlo Tree Search methods have given promising results on high dimensional stochastic planning problems [5] in continuous domains [6, 2]. Among the many potential applications, we consider the field of energy policies. The quality of energy policies has a huge financial and ecological impact. Also, resources being limited, power generation is relying increasingly on renewable energies. These energies (water stocks, solar panels, etc) are subject to high variations, and cannot be adjusted on demand. Thus, there is a growing need for a smart planning of how we use them. The current state of the art methods used in the industry mostly relies on approximations of the real problem, and fail to cope with an increase in the dimension of the state space. MTCS methods present the advantage of dealing with the exact problem. They also do not require any

expert knowledge about the problem itself, and have been known to be quite robust with respect to an increase in the dimension of the state space.

In [1], it is shown that, when using UCT-based algorithms in a domain where the transition is stochastic, it is important to control the number of different random outcomes to explore for each couple state/action. In the case of the Klondike game, their experiments show that it is better to explore about 5 different random outcomes per couple state/action, in comparison with exploring only 1 outcome, or 10 outcomes. Depending on the number of random outcomes explored, the performances of UCT-based algorithm vary significantly. The authors also suggest that a progressive widening technique could be interesting, instead of a fixed sampling width. This idea seems particularly relevant, especially in domains where the stochastic elements of the transition have a continuous support. Such technique has already been introduced in [2]. In this paper, the authors show how the original version of MCTS fails to converge towards the optimal solution in stochastic and continuous domains. They propose a solution to this issue, called Double Progressive Widening. Instead of limiting the number of explored random outcomes of each couple state/action, this technique progressively widens the tree at each node, according to the number of simulations of these nodes. However, this method still relies on two parameters, that remain to be tuned.

The first part of this paper (Section 2) will present the MCTS algorithm, as well as the Double Progressive Widening technique. We illustrate its efficiency on a simple stock problem, with a small stochastic part. The second part of this paper (Section 3) is devoted to the impact of the parameters of the Double Progressive Widening. We observe that, although the best set of parameters can vary across different problems, some settings are safe, and can guarantee reasonable performances, independently of the distribution of the random events.

2 Experiments on a basic stock problem

In this section, we study the impact of Double Progressive Widening (DPW) on the MCTS algorithm. We compare it to the standard Simple Progressive Widening MCTS (MCTS-SPW) on a basic stock problem. Let us first remind the MCTS algorithm, along with the double progressive widening technique, as introduced in [2].

2.1 MCTS with double progressive widening (DPW)

We consider a standard sequential decision making problem under uncertainty. The decision maker has to choose an action at every time step, until he reaches the horizon, where the reward is computed. We will note S the state space, D the decision space, and H the horizon. At each time step, the decision maker is given an initial state, and must return a decision. The goal of the decision maker is to find a policy that, given an initial state, returns the decision that maximizes the expected reward.

The MCTS algorithm explores some of the reachable states, by building a tree. Its nodes represent the states. The root of the tree is the initial state. The MCTS algorithm runs for a given time budget, and then returns a decision. The core of this algorithm lies in the way the tree is built. It aims at progressively exploring new states, by trying new decisions, while still exploiting the already explored states. The balance between exploration and exploitation is ensured by the Double Progressive Widening.

The core of the algorithm can be divided into four distinct phases, as shown in Fig. 1.

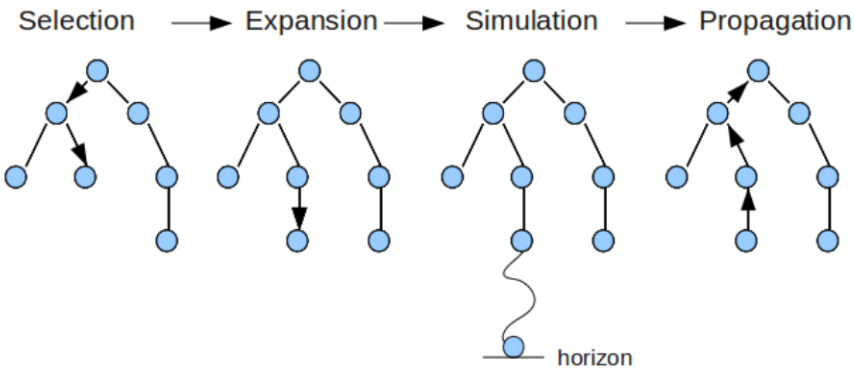

Fig. 1. MCTS algorithm phases.

First, we navigate through the already constructed nodes of the tree (selection phase). Then, when we decide to explore a new state, from the node that we reached, we try a new decision (expansion phase). Then, we simulate a series of decisions until we reach a final state (simulation phase). Finally, we compute the reward corresponding to the final state we just reached, and back propagate the information along the path that led to this state (propagation phase).

The Progressive Widening is the technique that defines when we decide to end the selection phase. More precisely, it defines when we decide to explore new states (i.e. add nodes to the tree), and when we decide to exploit known states.

The basic version of MCTS, as seen in [4], will be reffered to as MCTS with Simple Progressive Widening (MCTS-SPW). This method is suitable to deterministic problems. As we will see, it is inefficient on stochastic problems, even when the random part has a very small impact on the reward function.

What follows, is the formal description of MCTSwith Double Progressive Widening (MCTS-DPW), as seen in [2].

Double Progressive Widening (DPW) applied in state s with constants $C > 0$, $\alpha \in]0,1[$, and $\beta \in]0,1[$.
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:
$totalReward_t(s,o) = \sum_{1 \leq l \leq t-1, o_l(s)=o} r_l(s)$
$nb_t(s,o) = \sum_{1 \leq l \leq t-1, o_{(l)}(s)=o} 1$
$score_t(s,o) = \frac{totalReward_t(s,o)}{nb_t(s,o)+1} + k_{ucb}\sqrt{\log(t)/(nb_t(s,o)+1)}$ $(+\infty$ if $nb_t(o) = 0)$
Let $k' = \lceil Cnb_t(s, o_{(t)}(s))^{\beta} \rceil$
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.

2.2 Energy stock problem with low stochasticity

We first experiment MCTS on a basic energy stock management problem, defined as follows. There are N stocks, and at every time step, we have to decide how

much energy is taken from each stock. These stocks can be water reservoirs, and can be connected to each other. We consider the case where they lay in a valley: the water taken out of the first stock goes into the second, and so on, and the water taken out of the last stock is lost in the ocean. There is also a thermal plant, that has a given maximum production capacity. Producing energy from the water stocks is free (although the stocks are finite), and using the thermal plant has a cost, quadratic in the amount of energy produced. At each time step, each stock receives an extra inflow (from the rain), that follows an unknown random distribution. The goal of the decision maker is to satisfy a time varying demand, at the lowest possible cost.

In this first section, we consider the case where the inflows are small, relatively to the stocks and the demand. It is important to note that, since the MCTS does not measure any distances, having a relatively small random part in our problem does not help much the algorithm.

As we will see, even adding an infinitesimal random part in the transition, as long as the support of the distribution is continuous, causes a lot of trouble to a basic version of MCTS.

Our experiment was made on a problem with 2 stocks and 6 time steps. The initial stock level was set to 100. The thermal plant had a maximum production capacity of 50 per time step. The average demand per time step was of 75 (it varies over time, to reflect the seasonality of the energy consumption). The inflows on each stock were uniformly distributed on $[0, 1]$.

We compared the performances of four algorithms. First, the original version of MCTS in continuous domains (MCTS-SPW); basically as in [4] (for a problem with no stochastic part), i.e. with a bandit modified as in [7]. The second, third and fourth algorithms are three different versions of the MCTS with double progressive widening (MCTS-DPW[2]); each one with a different value for the parameter β. The value of α was fixed to 0.4 throughout the experiment. This choice of α is traditional. As we will see in this paper, this is a reasonable choice.

As we can see in Fig. 2, MCTS-SPW performs quite poorly. Indeed, the support of the distribution of the inflows being continuous, the probability of drawing the same random event twice is null. Thus, this algorithm builds a tree of depth one, and never explores the upper levels except through Monte Carlo simulations. Doing so, it tends to evaluates decisions as if the policy used after these decisions was merely random walk.

This is obviously not the right way to evaluate a decision. Indeed, if we make a decision at time step t, when reaching a new state, at time step $t + 1$, we will be given another time budget to make a new decision. This will be repeated until we reach the horizon. Hopefully, the decisions made from $t + 1$ to H will be better than the result of a pure blind search.

To account for this, we need to bias the evaluation of the new states, so that this evaluation simulates an intelligent agent, and not a random walk. This is exactly the aim of DPW.

The MTCS-DPW with $\beta = 0.75$ performs similarly to MCTS-SPW. This is due to the fact that the value of β leads MCTS to build a very wide tree,

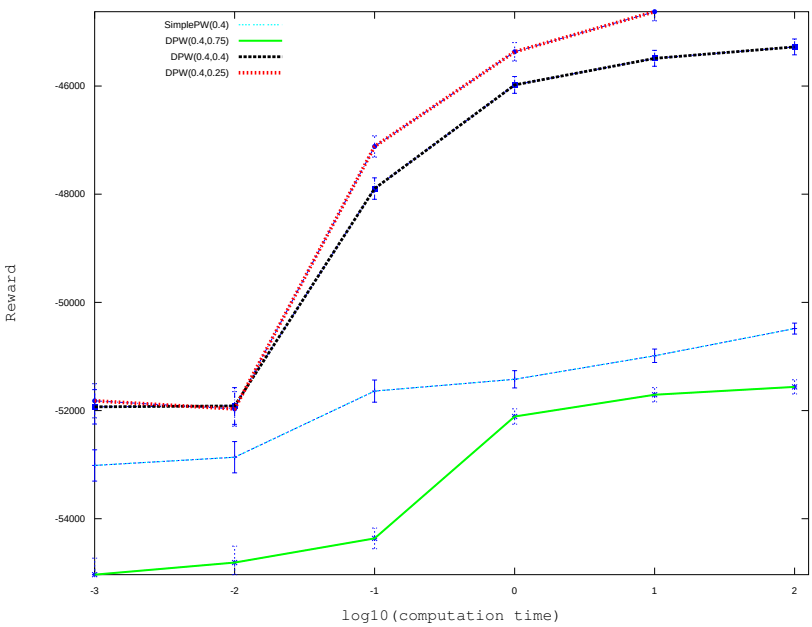

Fig. 2. 2 stocks, 6 time steps, small inflows.

not tall enough. As MCTS-SPW, it explores the upper levels of the tree mostly through Monte Carlo simulations. Incidentally, this reduces the average number of simulations per node. The efficiency of MCTS relies on statistical information obtained through a large number of simulations. Hence, MCTS performances decrease with the average number of simulations per node. The MCTS-DPW with $\beta = 0.4$ and 0.25 perform much better than the two first algorithm, with the fastest convergence rate going to the version with $\beta = 0.25$.

This is not a surprising result, given the setting of the problem. Since the random inflows are very small compared to the other parameters of the problem, it is of little importance to build a very wide tree at the random node level (i.e. to explore many different random occurrences of the events). In this setting, the best strategy is to explore very few random occurrences, and to build a tall tree, with many leaves being final nodes.

3 Experiments on a problem with high stochasticity

In this section, we add a discrete random event to the transition. We want this event to have a low probability of happening, but significantely impact the reward.

There are two main reasons for us to study this kind of problem. First, discriminate methods that see rare random events ahead in time and those that do not. Second, this kind of problem is often observed in real world data. In the models used by industrials for building energy policies, most assets (thermal plants, etc) have a low probability of failure. If these rare events are not taken into consideration, one may take unwise risks.

3.1 The discriminating power of the stock problem with thermal plant failure

To see how this can easily discriminate methods that misses rare events, let us consider the following modified stock problem. The stocks are tight, i.e. the total amount of energy that can be produced out of the stocks is lower than the expected demand. The decision maker is forced to use the thermal plant at least once to satisfy the demand. The thermal plant has now a probability of failure p at the last time step. With probability p, the demand at the last time step has to be satisfied with the stocks only; if not, the decision maker has to pay a significant failure cost.

In this situation, a method that does not see the possibility of a thermal plant failure will use the water stocks as much as possible, to lower the quadratic thermal production cost. More precisely, the thermal cost being quadratic, such a method will produce exactly the same amount of energy out of the thermal plant at each time step. Doing so, the decision maker will be left, at the last time step, with exactly the stock required to match the demand with the help of the thermal plant, expected to produce the same amount of energy as in the previous timestep. In the case where the failure does not happen, this greedy strategy returns the lowest possible cost. But, in the case of failure, the decision maker has not enough stock left, and pays the very high failure cost.

To effectively discriminate methods that does not see the rare event, we make sure that the expected cost of a failure is higher than the expected benefits of a greedy strategy.

In such a problem, a method that does see the possibility of a thermal plant failure will save some water to make sure that the decision maker has enough stock at the last time step to satisfy the demand without the thermal plant. The thermal cost being quadratic, in the case where the failure does not happen, this is more costly than the greedy stratefy. However, in the case where the failure does happen, the decision maker can satisfy the demand, without paying the failure costs.

Note that the practitionners problem is much more complicated and subtle than this one. They have to evaluate the cost of a failure (what is the cost of shutting down electricity in a certain region for a certain duration?), and to evaluate the probability for this event to happen. In this perspective, we want to inquire on how, by modifying the double progressive widening parameters, we can obtain different versions of MCTS. One may then choose a different set of parameters, according to the type of problem he/she is facing (low, or very low probability of failure, medium or high failure cost, etc).

3.2 Experiments with a low thermal failure probability

In this section, we consider the stock problem with thermal plant failure, with a short time horizon. We have 2 stocks, and a time horizon of 3. The initial stocks are of 100. The thermal plant can produce up to 50 at a cost 10 per square unit. The demand is time varying, but has an average value of 125. The failure cost is set to 100000 per missed units. The probability of failure is set to 0.1.

The parameter β being the one directly related to how we manage the exploration/eploitation dilemna at the random level, we first fix α to four different values, and study the impact of variations on β. Doing so, we observe that best results are obtained for $\alpha = 0.6$, for any value of β. The results for $\alpha = 0.6$ are shown in 3. We observe that the best value for β is 0.6, followed very closely by $\beta = 0.8$. On Fig. 4, we fixed β to 0.6, and plotted the performances of MCTS with different values of α. In this figure, we see that for a given value of β, the results obtained with different values of α can vary by a ratio of one to four.

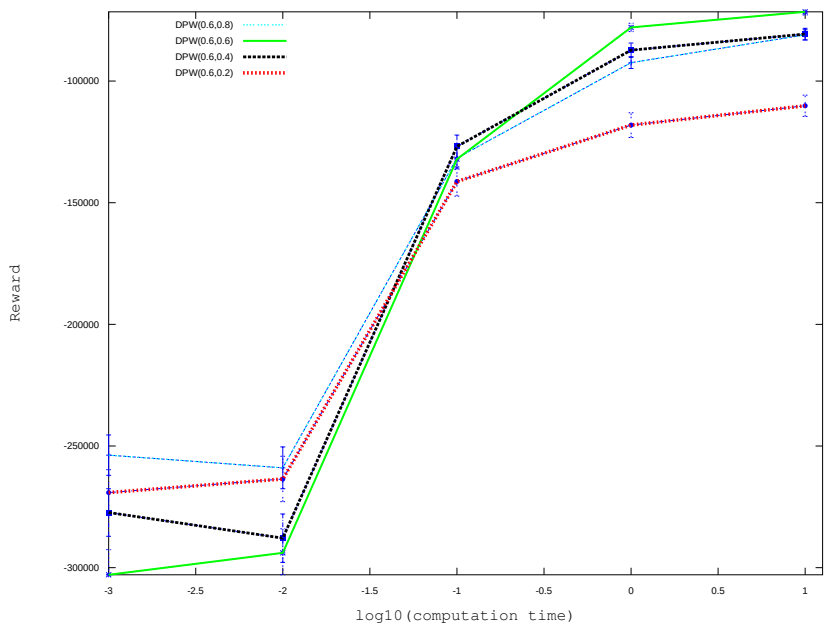

Fig. 3. 2 stocks, 3 time steps, α=0.6

From these figures, we can conclude that: First, the best overall result is obtained with the version having $\alpha = \beta = 0.6$. Second, a poor setting of α (e.g. to 0.2) makes the different versions of MCTS equivalent, all of them giving

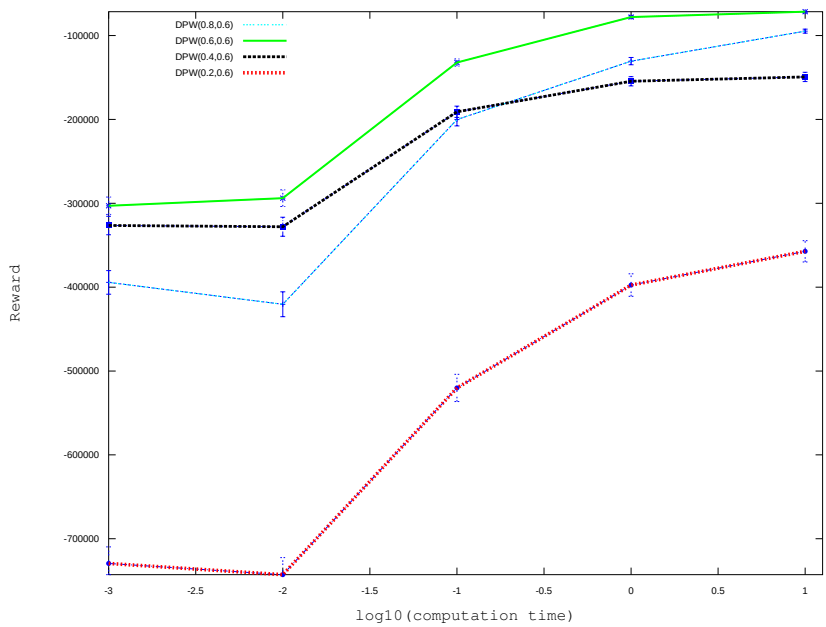

Fig. 4. 2 stocks, 3 time steps, β=0.6

significantly worst results than the other settings of α. A wise tuning of α seems to be necessary before even trying to tune β.

In this section, we saw how the addition of a low probability/high impact event to the problem can dramatically change the values of the best set of parameters. More precisely, the best value for β went from 0.2 to 0.6. This shows that, the larger the impact of the random events is, the higher the value of β should be.

3.3 Experiments with a higher thermal failure probability

In this section, we change the probability of thermal plant failure. We set it to 0.5, making it a relatively high probable event. We keep the same failure cost, so that this event still has a very high impact of the overall reward. In this setting, the event of a failure is easier to spot. One intuition would be to guess that, in this case, one does not want to explore a lot of random occurrence, and focus on exploiting a few of them.

We ran the experiment, and it turned out to be wrong. We obtain similar results to the case with a low probability. The best setting for α remains around 0.6. The setting of α is still crucial, as is shown in Fig. 5. By plotting the results with α set to 0.6, we observe that the best values for β are 0.6 and 0.8. Note

that the rewards are lower than in the previous section: this is due to the higher probability of thermal plant failure, that makes the problem much harder overall.

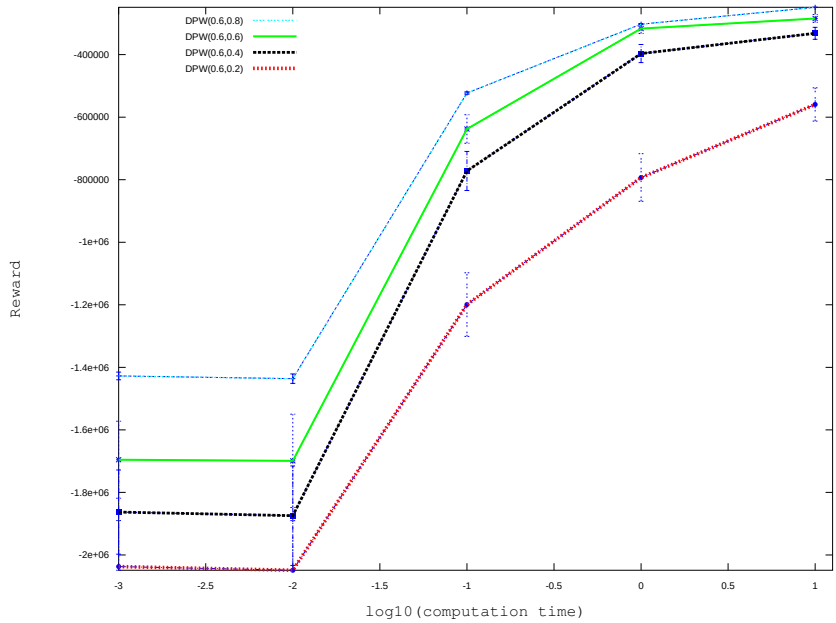

Fig. 5. 2 stocks, 3 time steps, α=0.6, $p_{fail} = 0.5$

These results indicates that, as long as a random event has a very high impact on the reward function, one wants to have a fine perception of it. To do so, one needs to use a sufficiently high value for the parameter β.

4 Conclusion

We obtained experimental evidence that, to solve even an easy stochastic stock management problem, MCTS-PW outperforms MCTS-SW, assuming a reasonable and easy to find set of parameters is used. We also saw that to solve a problem largely driven by its stochastic part (i.e. thermal plant failure), one needs to carefuly choose the second parameter of DPW (i.e. β). Our results indicate that a safe choice for β is between 0.4 and 0.6. Indeed, setting it to 0.8 gives very poor results in case of a low impact of the stochastic part, and setting it to 0.2 is inefficient on problems with high stochasticity. This choice gives a version of MCTS that performs well in both cases, with low and high stochasticity, independantly of the probability of the high impact events. One should

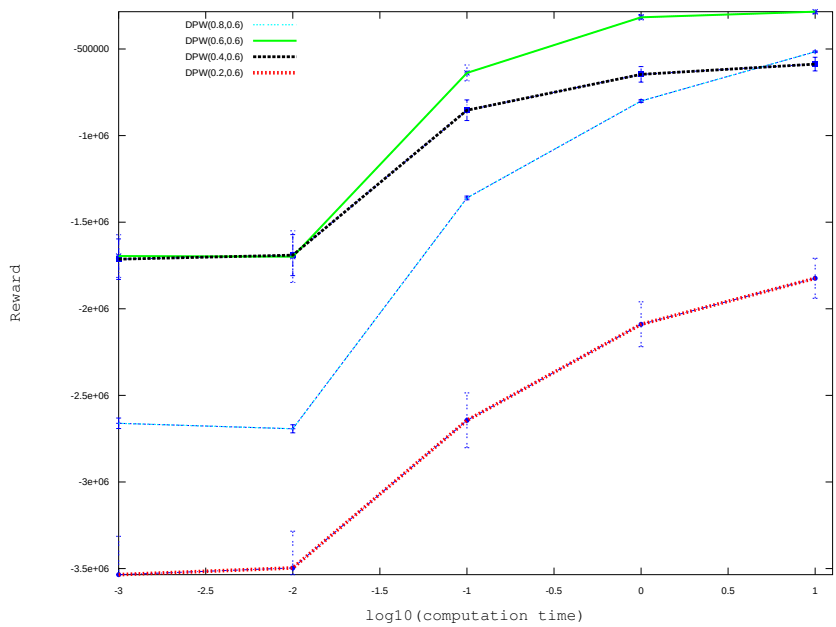

Fig. 6. 2 stocks, 3 time steps, β=0.6, $p_{fail} = 0.5$

keep in mind though, that the relative scale of the random events, compared to the deterministic part of the problem, does matter when chosing the parameter β.

Our study also confirmed that the first parameter of the progressive widening, α, is crucial to obtain reasonable performance. Our results indicates that one should set it to a value between 0.4 and 0.6.

As future work, we would like to develop a simple adaptative tuning of β, to make the algorithm robust to variations in the scale of the random events.

5 Acknowledgments

We thank Grid5000, that made our experiments possible (`https://www.grid5000.fr/mediawiki/index.php/Grid5000:Home`).

References

1. R. Bjarnason, A. Fern, and P. Tadepalli. Lower Bounding Klondike Solitaire with Monte-Carlo Planning. In *ICAPS'09*, 2009.

2. A. Couetoux, J.-B. Hoock, N. Sokolovska, O. Teytaud, and N. Bonnard. Continuous Upper Confidence Trees. In *LION'11: Proceedings of the 5th International Conference on Learning and Intelligent OptimizatioN*, page TBA, Italie, Jan. 2011.
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. Nakhost and M. Müller. Monte-carlo exploration for deterministic planning. In C. Boutilier, editor, *IJCAI*, pages 1766–1771, 2009.
6. 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.
7. Y. Wang, J.-Y. Audibert, and R. Munos. Algorithms for infinitely many-armed bandits. In *Advances in Neural Information Processing Systems*, volume 21, 2008.