Improving the exploration in Upper Confidence Trees

Adrien Couëtoux[1,2,3], Hassen Doghmen[1], and Olivier Teytaud[1,2]

[1] TAO-INRIA, LRI, CNRS UMR 8623,
Université Paris-Sud, Orsay, France
[2] OASE Lab, National University of Tainan, Taiwan
[3] Artelys, 12 rue du Quatre Septembre Paris, France

Abstract. In the standard version of the UCT algorithm, in the case of a continuous set of decisions, the exploration of new decisions is done through blind search. This can lead to very inefficient exploration, particularly in the case of large dimension problems, which often happens in energy management problems, for instance. In an attempt to use the information gathered through past simulations to better explore new decisions, we propose a method named Blind Value (BV). It only requires the access to a function that randomly draws feasible decisions. We also implement it and compare it to the original version of continuous UCT. Our results show that it gives a significant increase in convergence speed, in dimensions 12 and 80.

1 Introduction and motivation

We consider a high dimensional continuous and stochastic sequential decision making problem. Both the decision space and state space are continuous. For the sake of simplicity, both have the same dimension N. There is a finite time horizon H, after which no further decisions are made. At each time step $t < H$, given a current state s_t, the optimizer has to make a decision d. In this paper, we will denote the set of feasible decisions from state s as $X(s)$. We will also denote the set of explored decisions from state s after the n^{th} iteration, $n \geq 1$, as $D_n(s)$.

We consider that the optimizer has at its disposal a model with a transition function f and a sampling function φ. The transition function takes as inputs a state s_t and a decision $x \in X(s_t)$. Its outputs are a state s_{t+1} and a reward $r_{t+1} \in \mathbb{R}$. The sampling function takes as input a state s_t, and its output is a decision $x \in X(s_t)$. The optimizer has no other knowledge of the model than these two functions. Both functions can be (and are, in our experiments) stochastic: $f(s_t, d)$ and $\varphi(s_t)$ are two multidimensional real random variables, their probability distributions being unknown to the optimizer.

The optimizer is given an initial state s_0, and its objective is to maximize the accumulated reward $\sum_{1 \leq i \leq H} r(i)$.

In this context, UCT like algorithms have been some of the most efficient methods, like MCTS in the game of Go [6, 4, 7, 8], or continuous MCTS on energy

management problems [5]. This is due, mostly, to the fact that without any dimension reduction technique (that is application specific), all other well known methods, like dynamic programming, fail [1, 2].

However, in its current form, continuous MCTS does not use any information when it explores new decisions. This is not such a dramatic issue in discrete cases, or in low dimensional continuous cases, as one can just blindly cover most of the search space. But, in the case of a high dimensional continuous decision space, we think that the convergence speed could be greatly increased by biasing the way new decisions are explored. As mentioned before, we consider the case where the sampling of feasible decisions is "black box". This means that the set of feasible decision, as well as the inside of the sampling function φ, are unknown to the optimizer. This keeps biased sampling methods out of our options. We chose to consider this case because, to have access to the inside of the sampling function, one needs to know precisely the constraints of the model, and to implement them. Not only can this work take enourmous amounts of time (i.e.,hundreds of man hours), but the resulting feasible space may also be highly non convex, resulting in even more work to develop sampling functions on these spaces. In short, biased sampling, in our opinion, should be relevant for very problem-specific methods.

One could also count on an approximate knowledge of the feasible space, sample on this approximation, and apply a large penality to infeasible decisions. This is a valid option for many problems, but, in energy management problems (our main current application of interest), the feasible decisions are extremely sparse in the convex envelop, making this approach less interesting.

Our approach is focused on using the very limited information of the transition and sampling functions, in combination with the information progressively made available during the course of the simulations.

This is inspired by a work on continuous and stochastic bandits problems [3], which provides a method that ensures that no area of the feasible decisions set is left unexplored, while focusing on promising areas of this set. However, it focuses on the theoretical aspect of the problem, and requires some assumptions that often do not hold in our case. In particular, it requires the knowledge of the set of feasible decisions. Still, our approach follows the same ideas: explore the empty areas first, and then to focus on areas where promising decisions are.

We first quickly review the state of the art form of MCTS in a continuous setting. Then, we introduce its new variant termed MCTS with Blind Value (MCTS-BV). Then, we show some experimental results on an energy management problem, where we compare the two versions, first in a small dimension setting, then in a larger dimension setting ($N = 80$). Finally, we present some experimental results on the tuning of one parameter of MCTS-BV.

2 State of the art of continuous Upper Confidence Trees

The UCT algorithm builds a tree where the nodes represent the reachable states, and the arcs the feasible decisions. By progressively adding arcs and nodes from its root (that represents the initial state given to the optimizer), more infor-

mation is gathered. When the algorithm runs out of time, it selects the most promising decision reachable from the root (usually, the one that has been simulated the most). Among the crucial mechanisms in this algorithm, there are: when to add a new decision to the tree or when to exploit a known decision, and how to select a known decision once we have chosen not to add a new one. The first part is usually dealt with by using Progressive Widening [5], while the second is usually dealt with by using an Upper Confidence Bound formula (UCB).

For more detailed information about the current state of the art form of MCTS with Double Progressive Widening (MCTS-DPW), please read [5].

3 Blind Value

The principle of Blind Value is to help the exploration of new decisions. One can want to explore a new decision from any state already in the tree. Although in our case, the optimizer cannot bias the sampling of new decisions, we propose a method that does use the information available in the tree. More precisely, we use the information about the children of the current node. In terms of states and decisions, it means: when we want to explore a new decision from state s, we use information about all the decisions explored from this state s in the past simulations to select a new decision $x \in X(s)$ to be explored from state s.

Note that one could use any other information in the tree: brother nodes, grand children nodes, father node, etc. However, even the exploitation of the direct children of a node only is computationally costly. And, the more distant in the tree some information is, the more likely it is to be irrelevant to the node we are currently in (states might be very different, and this type of problem is also highly time step dependant). [8] has proposed the use of Rapid Action Value Estimates (RAVE), which are an interesting other possibility; we will consider the mixing of blind value with RAVE values in a further work. [8] also proposed the use of information from related nodes; after preliminary positive results, this was later removed from the corresponding implementations (for the game of Go) for correctly tuned implementations.

The idea of BV is to try to explore decisions that are far away from known decisions during the first simulations, and then to focus on areas that have a lot of decisions with high UCB values. This is done by sampling a number of new decisions, and by selecting one of them according to a combination of these two criterions (explore unknown regions and explore regions with many decisions with high UCB values in it).

More precisely, we sample a number $M \geq 1$ of random decisions, and we pick the one that is the most interesting to explore. The way we measure the interest of a decision is through a function from the decision space to $\mathbb{R}$, denoted $BV(.)$ (Blind Value). This function can be defined in many different ways. In this paper, at an iteration n, given a state s and a decision $x \in X(s)$, we chose to define $BV(x)$ as the minimum over $D_n(s)$ of the sum of two parts. The first part is the UCB value of $d \in D_n(s)$, the second is the distance between $x \in X(s)$

and $d \in D_n(s)$, multiplied by an adaptation coefficient. We use the standard euclidian distance, but any other distance could be used instead.

More precisely, the Blind Value of x in state s with decisions $d \in D_n(s)$ already explored is

$$BV(x) = \min_{d \in D_n(s)} UCB(d) + \rho dist(d, y).$$

What follows is the detailed blind value algorithm:

Exploration of new decisions
Input: a state s, a set D of already explored decisions, an integer M, and a distance function over the decision space, *dist*
Output: an unexplored decision x.
Generate M random decisions. Let X be the set composed of these decisions.
Compute $a = UnbiasedStandardDeviation_{d \in D}(UCB(d))$
Compute $b = UnbiasedStandardDeviation_{x \in X}(dist(x, 0))$, 0 being the center of the domain
Compute $\rho = \frac{a}{b}$
return $x = argmax_{y \in X} BV(y, \rho, D)$

Computing BV (Blind Value)
Input: an unexplored decision y, a real number ρ, and D the set of explored decisions
Output: a real number $BV(y, \rho, D)$.
return $min_{d \in D}(\rho \times dist(d, y) + UCB(d))$

4 Experimental comparison

Our test case is an energy management problem. There are N energy stocks, H time steps, and a thermal power plant with a given maximum capacity and production cost function. In our experiments, we used a quadratic cost function. At each time step, each stock also receives an inflow. Each inflow follows its own independent random distribution.

At each time step, the decision maker has to decide how much to produce from each stock, and how much to produce from the thermal plant. His goal is to satisfy a time varying demand at the lowest possible cost.

We ran two algorithms on this problem: the continuous version of MCTS, as introduced in [5], and the same algorithm with the addition of Blind Value (MCTS-BV), with the sample size parameter set to 20. This experiment was run with 12 stocks and 16 time steps. The results are shown in fig 1 (left). In this experiment, as in the following ones, each point is computed from 10000 runs of the algorithm on one problem instance. The 95% confidence intervals are plotted as blue segments around the points, even though their small size can make them very hard to see in some cases.

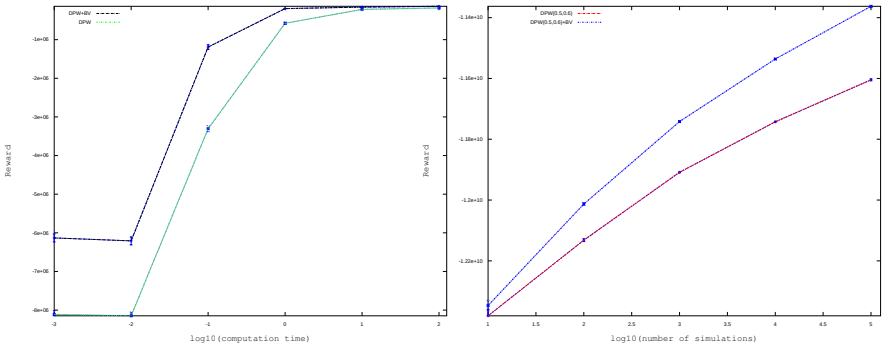

Fig. 1. Left: reward, as a function of the computation time. Problem settings: 12 stocks, 16 time steps. MTCS with BV is 10 times faster than MCTS, for budgets up to 10 seconds per decision. Right: reward, as a function of the number of simulations per decision. Problem settings: 80 stocks, 6 time steps. MTCS with BV is 10 times faster than MCTS, for all budgets.

This figure shows that even in dimension 12, BV already gives an edge of magnitude 10 to MCTS, in terms of computation time (to reach a certain level of performance, MCTS requires 10 times as many simulations as MCTS-BV). The problem being reasonably easy, we also see that this edge decreases when the computation time increases (but only for a computation time of about 3 minutes). This is due to the fact that, as the budget gets bigger, both algorithms get very close to the optimum, in terms of reward.

This is why we made a second experiment, on the same problem, with a much higher dimension. In this experiment, there are 80 stocks, 6 time steps, and $M = 640$. With the information given to the algorithms (just the transition and the sampling functions), this problem is incredibly difficult, and naturally has very low reward (the highest average possible reward being around -2.5×10^7). Given its dimension, there is no way of exploring the entire decision space, even with a very low density. The results are shown in Fig. 1 (right).

This figure shows that BV still gives an edge of magnitude 10 to MCTS in terms of computation time, even though we were not able to approach the optimum with our computing capacities. One can also note that in this setting, the difference seems to be increasing as the budget increases. This leads to think that on very difficult problems, BV can divide by ten, or even more, the computing time necessary to reach a certain level of performance.

5 Conclusion and future work

We introduced a new variant of continuous MCTS to better solve high dimensional problems. Our experimental results show that this variant, MCTS-BV,

improves the original algorithm by a factor 10 (it reaches the same level of performance with 10 times less simulations than the original). This result holds for even relatively small dimension problems (dimension 12). The edge given by BV seems to be even bigger in very high dimensions, but this could be confirmed by longer experiments.

We believe that, especially in its simplest form, Blind Value is very easy to implement, and can significantly increase the convergence speed of MCTS on continuous and stochastic planning problems.

It could also be coupled with other ways of exploiting information throughout the tree. One of the promising leads is the use of RAVE values, first introduced in the game of Go [8], and currently extended to continuous domains. While Blind Value only uses the decisions at the current level in the tree (horizontal propagation), RAVE propagates information through the path of each simulation, from child to father node (vertical propagation) enabling the algorithm to attribute a value to decisions not yet explored from one specific state.

Our future work will focus on setting an adaptive form for the sampling size parameter, to adapt it to the time budget. We also plan on trying different variants of the formula used to rank the decisions in the pool, by changing the distance, for example.

Acknowledgments. We thank Grid5000, that made our experiments possible (`www.grid5000.fr`). We also thank the National Science Council of Taiwan for grants NSC97-2221-E-024-011-MY2 and NSC 99-2923-E-024-003-MY3.

References

1. R. Bellman. *Dynamic Programming.* Princeton Univ. Press, 1957.
2. D. Bertsekas and J. Tsitsiklis. *Neuro-dynamic Programming.* Athena Scientific, 1996.
3. S. Bubeck, R. Munos, G. Stoltz, and C. Szepesvri. Online optimization in x-armed bandits. In *In Advances in Neural Information Processing Systems 22*, 2008.
4. 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.
5. 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.
6. 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*, pages 72–83, 2006.
7. R. Coulom. Computing elo ratings of move patterns in the game of go. In *Computer Games Workshop, Amsterdam, The Netherlands*, 2007.
8. 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.