Continuous Rapid Action Value Estimates

Adrien Couetoux, Mario Milone, Matyas Brendel, Hassen Doghmen, Michèle Sebag, Olivier Teytaud

Continuous Rapid Action Value Estimates

Adrien Couëtoux Mario Milone, Mátyás Brendel,
Hassen Doghmen, Michèle Sebag
TAO, INRIA-CNRS-LRI, Univ. Paris-Sud, F-91405 Orsay
Olivier Teytaud
TAO, INRIA-CNRS-LRI, Univ. Paris-Sud, F-91405 Orsay
OASE Lab, National University of Tainan, Taiwan

November 23, 2011

Abstract

In the last decade, Monte-Carlo Tree Search (MCTS) has revolutionized the domain of large-scale Markov Decision Process problems. MCTS most often uses the Upper Confidence Tree algorithm to handle the exploration *versus* exploitation trade-off, while a few heuristics are used to guide the exploration in large search spaces. Among these heuristics is *Rapid Action Value Estimate* (RAVE). This paper is concerned with extending the RAVE heuristics to continuous action and state spaces. The approach is experimentally validated on two artificial benchmark problems: the treasure hunt game, and a real-world energy management problem.

Keywords: Rapid Action-Value Estimates, continuous domains, reinforcement learning

1 Introduction

After [13] reinforcement learning problems are most generally formalized using the Markov Decision Process (MDP) setting. An MDP is characterized by its state space $\mathcal{S}$, action space $\mathcal{A}$, the environment dynamics described from the probability $P^a_{ss'}$ of going to state s' upon executing action a in state s, and the associated reward function $R^a_{ss'}$.

Pioneered in the context of computer-Go [7], Monte-Carlo Tree Search (MCTS) algorithms have been used to successfully tackle large-scale MDPs (section 2). In particular MCTS accommodates large-scale generative settings, where the transition function is known through a simulator as opposed to being available in closed form. MCTS most often combines the Upper Confidence Tree (UCT) algorithm first introduced by [8] and a few generic heuristics. UCT addresses the exploration *versus* exploitation trade-off in each state of the tree search space through extending the Upper Confidence Bound algorithm proposed by [2] in the Multi-Armed Bandit setting. Notably, the MCTS/UCT has

revolutionized quite a few large-scale RL problems, ranging from computer-Go [7] to expensive optimization [12, 6] and planning [11].

The main two heuristics combined with UCT aim at guiding the exploration strategy, through limiting the number of considered actions with *Progressive Widening* (PW) [5, 4, 16], and selecting the most promising actions with *Rapid Action Value Estimate* (RAVE). For the sake of self-containedness, section 2 introduces Monte-Carlo Tree Search and the PW and RAVE heuristics.

While RAVE is acknowledged to be a key factor of MCTS efficiency, to our best knowledge it has been limited until now to discrete action and state spaces. Motivated by applications in management and robotics, this paper focuses on extending RAVE to continuous action and state spaces using a Gaussian convolution-based smoothing (section 3). The proposed approach is experimentally validated on two problems, the artificial treasure hunt benchmark, and a real-world energy management problem (section 4). The paper concludes with a discussion and some perspectives for further research.

2 Monte-Carlo Tree Search

This section assumes the reader's familiarity with the Reinforcement Learning framework, referring to [13] for the standard notations. In the following, $Q^\pi(s,a)$ denotes the value function i.e. the expected reward gathered when executing action a in state s and following policy π for all subsequent actions until arriving at a terminal state:

$$Q^\pi(s,a) = E_\pi \left[\sum_{h=0}^{s_{h+1}\text{terminal state}} P^{a_h}_{s_h\, s_{h+1}} R^{a_h}_{s_h\, s_{h+1}} \mid s_0 = s, a_0 = a, a_h = \pi(s_h) \right]$$

Monte-Carlo Tree Search proceeds by estimating the value function through averaging the empirical cumulated reward along tree-walks, where each tree-walk starts in the initial node and follows the Upper Confidence Tree algorithm (section 2.1) until arriving in a terminal node.

Sections 2.2 and 2.3 thereafter respectively introduce the UCT algorithm and the PW and RAVE heuristics.

2.1 Upper Confidence Tree

In the baseline Monte-Carlo Tree Search, the action executed in each state is uniformly selected in the action space. Such a uniform sampling however does not enforce a good exploration *versus* exploitation trade-off; it is poorly effective when dealing with a large action space and/or a long time horizon.

Upper Confidence Tree (Algorithm 1), one of the best MCTS variants, does enforce an optimal exploration *versus* exploitation trade-off through the famed Upper Confidence Bound (UCB) [2]. Formally, UCB was devised for the multi-armed bandit setting. When considering a set of k arms (action nodes), letting n_i and μ_i respectively denote the number of times the i-th arm has been visited

and the empirical average reward then collected, UCB selects the arm i^* such that

$$i^* = \text{argmax} \quad \left\{ \mu_i + C\sqrt{\frac{\log \sum_{j=1}^{k} n_j}{n_i}}, i = 1 \dots k \right\} \tag{1}$$

where C is a problem-dependent parameter. Accordingly, the UCT-based policy noted π_{UCT} selects in each node s the action a^* defined as follows:

$$\pi_{UCT}(s) = \text{argmax} \left\{ Q^{\oplus}_{UCT}(s,a) = Q_{UCT}(s,a) + C\sqrt{\frac{\log n(s)}{n(s,a)}}, a \in \mathcal{A} \right\} \tag{2}$$

with $n(s)$ the total number of times state s has been visited, $n(s,a)$ the number of times action a has been selected in state s, and $Q_{UCT}(s,a)$ the empirical cumulative reward averaged over all times action a has been selected in state s. According to Eq. (1), every possible action must be selected once in each state, which is hardly tractable when the number of arms is large in front of the time horizon; likewise, Eq. (1) cannot be used for a continuous arm space. To address this limitation, the number of arms to be considered in each node tree is restricted (PW heuristics), and the choice of arms is controlled too (RAVE).

2.2 Progressive Widening

First introduced by [5], the Progressive Widening (PW) heuristics limits the number of considered actions in state s depending on the number $n(s)$ of times s has been visited. Progressive widening has also been used by [4] for continuous action spaces.

Specifically, the number $pw(n(s))$ of actions allowed by PW in state s is set to the integer part of $n(s)^{\frac{1}{p}}$, with $p = 2$ or 4; the interested reader is referred to [16] for a theoretical analysis of PW. Upon incrementing $pw(n(s))$, RAVE is used to select the next action to be considered.

2.3 Rapid Action Value Estimation

First pioneered in the context of computer-Go [7], Rapid Action Value Estimation (RAVE) aims at a more robust assessment of actions, through sharing the rewards gathered along different subtrees of the game tree. Formally, let $Q_{RAVE}(s,a)$ denote the empirical reward averaged over all tree-walks where action a has been selected after visiting state s, and let $m(s,a)$ be the number of such tree-walks. A variant of the UCT-policy (Eq. (2)) is defined as follows:

$$\pi_{RAVE}(s) = \text{argmax} \left\{ Q^{\oplus}_{RAVE}(s,a) = Q_{RAVE}(s,a) + C'\sqrt{\frac{\log m(s)}{m(s,a)}}, a \in \mathcal{A} \right\} \tag{3}$$

with $m(s)$ being the sum of $m(s,a)$ over all actions a.

Although taking more tree-walks into account contributes to a faster convergence of the action value estimate, $Q_{RAVE}(s,a)$ is a biased estimate of $Q(s,a)$,

Algorithm 1 The UCT algorithm, where $pw(.)$ is the progressive widening function (section 2.2)). Action $a_i(s)$ is uniformly drawn in $\mathcal{A}$ at the time $pw(n(s))$ is incremented.

UCT algorithm.
Input: a MDP, a state S, a time budget.
Output: an action a.
while time budget permits **do**
  $s = S$. // *starting a simulation, aka tree-walk*
  while s is not a terminal state **do**
    For all legal actions $a = a_1(s), \ldots, a_{pw(n(s))}(s)$
      Compute $Q^{\oplus}_{UCT}(s,a)$ // (Eq. 2)
    Select action a with maximal $Q^{\oplus}_{UCT}(s,a)$
    Let s' be the state reached from s when choosing action a.
    $s = s'$
  end while
  while s is not a terminal state **do**
    Select action a uniformly in $\mathcal{A}$ // *random episode*
    Let s' be the state reached from s when choosing action a.
    $s = s'$
  end while
  For all state action pair (s,a) visited during the tree-walk,
    Increment $n(s,a)$
    Update the average reward: $Q_{UCT}(s,a) \leftarrow Q_{UCT}(s,a) + \frac{1}{n(s,a)+1}[R(s) - Q_{UCT}(s,a)]$,
      where $R(s)$ is the cumulated reward from state s to the terminal state
end while
Return the action a which was simulated most often from S.

and should therefore be replaced by the true estimate $Q(s,a)$ whenever $n(s,a)$ permits to do so with reasonable confidence. It thus comes naturally to consider a dynamic weighted average of $Q_{RAVE}(s,a)$ and $Q(s,a)$, defining

$$\pi_{UR}(s) = \text{ argmax } \left\{Q^{\oplus}_{UR}(s,a), a \in \mathcal{A}\right\} \tag{4}$$

with

$$\begin{array}{rl} Q^{\oplus}_{UR}(s,a) = & \beta(s,a)Q^{\oplus}_{RAVE}(s,a) + (1-\beta(s,a))Q^{\oplus}_{UCT}(s,a) \\ \beta(s,a) = & \sqrt{\frac{k}{3n(s,a)+k}} \end{array} \tag{5}$$

where the *equivalence parameter* k represents the (domain-dependent) number of tree-walks required for the unbiased $Q_{UCT}(s,a)$ to provide as reliable an estimate as $Q_{RAVE}(s,a)$.

3 Continuous Rapid Action Value based Estimation

This section presents the proposed extension of RAVE to the case of continuous action spaces (section 3.1) and continuous space states (section 3.2). These extensions are discussed in section 3.3.

3.1 Continuous action spaces

While the presented discrete RAVE approach supports the fast estimation of action values, its reliability decreases as the number of actions which can be taken into account increases everything else being equal. Indeed in a continuous action space $\mathcal{A}$, the number of times a given action is tried is 0 in expectation, which renders RAVE useless.

It thus comes naturally to consider a smooth estimate of action values, e.g. using Gaussian convolution. Formally, given a training set $\mathcal{D} = \{(x_i, y_i), i = 1 \ldots n, x_i \in \mathbb{R}^d, y_i \in \mathbb{R}\}$, a Gaussian estimate of the value y associated to some $x \in \mathbb{R}^d$ is defined as

$$\hat{y}_\sigma(x) = \frac{1}{\sum_{i=1}^n e^{-\frac{1}{\sigma^2}d(x,x_i)^2}} \sum_{i=1}^n e^{-\frac{1}{\sigma^2}d(x,x_i)^2} \times y_i$$

where σ is a smoothing parameter weighting the relative importance of the nearest neighbors of x and $d(x,x')$ stands for the chosen distance on the space. In the remainder of this paper, only the Euclidean distance on $\mathbb{R}^d$ will be considered. In applications, prior knowledge about the application domain is provided through the choice of the distance.

Along this line, let $x_s = s.a_0 \ldots s_i.a_i \ldots$ denote a tree walk starting in s and let $\mathcal{R}(x_s)$ denote the associated cumulative empirical reward. $Q_{RAVE,a}(s,a)$ is

defined as:

$$Q_{RAVE,a}(s,a) = \frac{1}{\sum_{x_s,\ a_i\ in\ x_s} e^{-\log N_a \frac{d(a,a_i)^2}{\alpha_{action}}}} \sum_{x_s,\ a_i\ in\ x_s} e^{-\log N_a \frac{d(a,a_i)^2}{\alpha_{action}}} \times \mathcal{R}(x_s) \tag{6}$$

where α_{action} is a problem dependent parameter (proportional to the square dimension of the action space for the sake of homogeneity); N_a denotes the overall number of actions involved in all x_s, and the $\log N_a$ term is meant to peak the Gaussian convolution as the available empirical evidence increases. Counter $n(s,a)$ is likewise estimated using Gaussian convolutions and $\beta(s,a)$ is computed from $n(s,a)$ (Eq. (5)).

Both Q_{RAVE} and $Q_{RAVE,a}$ consider all tree-walks visiting state s and the cumulative reward gathered thereafter. The difference is that Q_{RAVE} only considers those tree-walks which have executed action a, whereas$Q_{RAVE,a}$ considers them all with a weight which decreases exponentially depending on the distance between the executed actions and the considered action a. As $Q_{RAVE,a}$ is even more biased than Q_{RAVE} (since it takes all actions into account, though weighted), one considers also the dynamic combination of Q_{RAVE} and $Q_{RAVE,a}$ as in Eq. (5)). One defines:

$$Q^{\oplus}_{UR_a}(s,a) = \beta(s,a)\, Q^{\oplus}_{RAVE,a}(s,a) + (1-\beta(s,a)) Q^{\oplus}_{UCT}(s,a)$$

and $\pi_{UR_a}(s)$ selects the action maximizing $Q^{\oplus}_{UR_a}(s,a)$.

Note that $Q_{RAVE,a}(s,a)$ is computed for a finite subset of $\mathcal{A}$ only, due to the progressive widening effects: only a finite number of actions is considered in each state node. The associated continuous rapid action value estimate is updated after each tree-walk.

3.2 Continuous state spaces

As already said, $Q_{RAVE,a}$ and Q_{RAVE} alike are strongly biased as they take into account every tree-walk conditionally to their visiting s and executing a or some similar action thereafter, although this action might be executed in a state s' very different from s.

In the case of continuous state spaces, it thus comes naturally to weight the contribution related to some state-action pair (s_i, a_i) depending on the distance between s and s_i. Formally, let us define $Q_{RAVE,a,s}(s,a) =$

$$\frac{1}{\sum_{x_s,\ s_i.a_i\ in\ x_s} e^{-\log N_{a,s}\left\{\frac{d(s,s_i)^2}{\alpha_{state}} + \frac{d(a,a_i)^2}{\alpha_{action}}\right\}}} \sum_{x_s,\ s_i.a_i\ in\ x_s} e^{-\log N_{a,s}\left\{\frac{d(s,s_i)^2}{\alpha_{state}} + \frac{d(a,a_i)^2}{\alpha_{action}}\right\}} \times \mathcal{R}(x_s) \tag{7}$$

As in Eq. 6, constant α_{state} is problem-dependent and proportional to the square dimension of state space, and $N_{a,s}$ is used to peak the Gaussian convolution as the available evidence to estimate $Q_{RAVE,a,s}$ increases.

3.3 Discussion

The proposed Continuous RAVE (cRAVE) heuristics involves two additional problem dependent parameters α_{action} and α_{space}, respectively involved in Eqs. (6) and (7). Note that $Q_{RAVE,a,s}$ can be viewed as a generalization of $Q_{RAVE,a}$(by taking $\alpha_{space} = \infty$), which itself generalizes Q_{RAVE} ($\alpha_{action} = \infty$).

Continuous RAVE can encapsulate prior knowledge on the action and space states, through using some informed dissimilarity function on the state and/or action spaces.

4 Experimental Validation

This section reports on the empirical validation of the Continuous RAVE heuristics, considering an artificial benchmark (section 4.2) and a real-world problem (section 4.3). The goals of experiments and experimental setting are first described.

4.1 Goals of experiment and experimental setting

The primary goal of experiments is to assess the efficiency of the action and (state, action) cRAVE heuristics, comparatively to the MCTS/UCT baseline. Both heuristics are plugged in the same MCTS/UCT algorithm with double progressive widening and default parameters [4]. After a few preliminary experiments, the value of the problem-dependent parameters α_{action} and α_{state} are set to d_{action} and $10^{-3}d_{state}$ where d_{action} and d_{state} respectively correspond to the dimension of the action and state spaces. The chosen distance in both action and state spaces is the Euclidean distance.

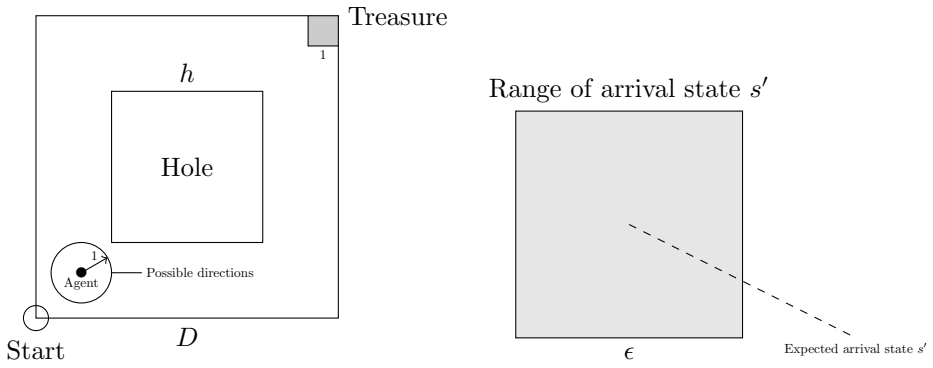
The equivalence parameter k (Eq. (5)) is set to 50.

In both problems the policy value is compared to the baseline approach for the same computational budget (number of tree-walks used to select an action, Alg. 2). Each value, averaged over independent runs, is reported together with the standard deviation.

The second goal of experiments is to study the sensitivity of the cRAVE heuristics with respect to the time horizon and size of the state space.

4.2 The TreasureHunt benchmark

The artificial treasure hunt problem involves a squared arena of size D (Fig. 1(a), left). The state space is $\mathcal{S} = [0, D]^2$. The goal of the agent, initially located in the lower left corner, is to reach the treasure in the upper right corner. The agent speed is fixed; its direction a varies in $\mathcal{A} = [0, 2\pi]$. In each time step, the agent gets an instant reward of -1; reaching the treasure location gets an instant reward of 1,000. Two options are considered: with deterministic and probabilistic transition probabilities; with and without hole (the square hole with size h is located in the center of the arena). Transition probabilities $P^a_{ss'}$ are defined as follows: upon selecting action (direction) a in state $s \in \mathbb{R}^2$,



(a) Treasure hunt problem: the agent must reach the treasure while avoiding the hole.

(b) Probabilistic transition model: the arrival state s' is perturbed by a 2D uniform noise.

Figure 1: The treasure hunt benchmark problem involves two options: the presence of a hole in the middle of the arena (left) and a probabilistic transition setting (right).

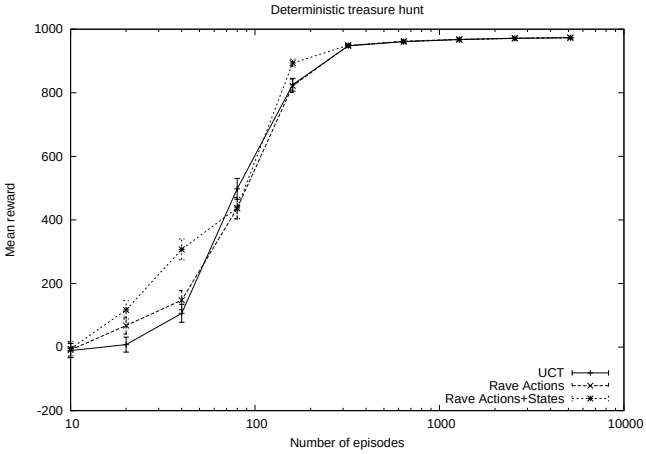
the agent arrives in state $s' = s + (cosa, sina) + (U[-\epsilon/2, \epsilon/2], U[-\epsilon/2, \epsilon/2])$, where $U[a, b]$ denotes a random variable uniformly drawn in $[a, b]$ ($\epsilon = 0$ in the deterministic case; Fig. 1(a), right). Being in the hole yields an instant reward of -500.

A tree-walk stops when the agent reaches the treasure, or falls in the hole, or after traveling a distance $10D$. In the deterministic setting, the optimal reward thus is 1,000 minus the shortest path between the starting location and the treasure (conditionally to avoiding the hole). Note that the optimal strategy in the probabilistic transition setting is not straightforward.

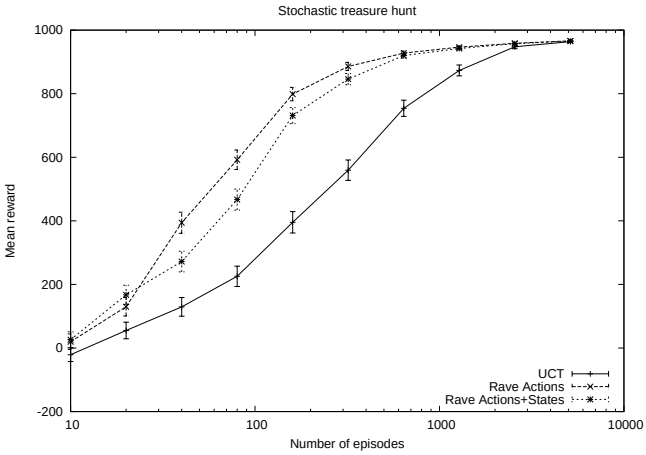
The motivations for the treasure hunt problem is to study the scalability of the cRAVE heuristics with respect to the size of the arena. It is worth mentioning that quite a few planning problems (path planning) can be formulated as treasure hunt problems in high dimensional spaces involving many holes (see e.g. [15]).

Figure 2(a) (top) displays the comparative results obtained by cRAVE$_{action,state}$, cRAVE$_{action}$ and UCT in the deterministic transition setting with no hole. In this most simple setting, there is no significant difference although cRAVE$_{action,state}$ significantly improves on UCT for small time budgets. Interestingly, cRAVE$_{action,state}$ does not much improve on cRAVE$_{action}$. This is explained as the optimal trajectory is the straight line from the initial state to the treasure location: the optimal action does not depend on the current state in this simple problem. The advantage of cRAVE$_{action,state}$ will become significant in more complex settings when the optimal decision depends on the current state.

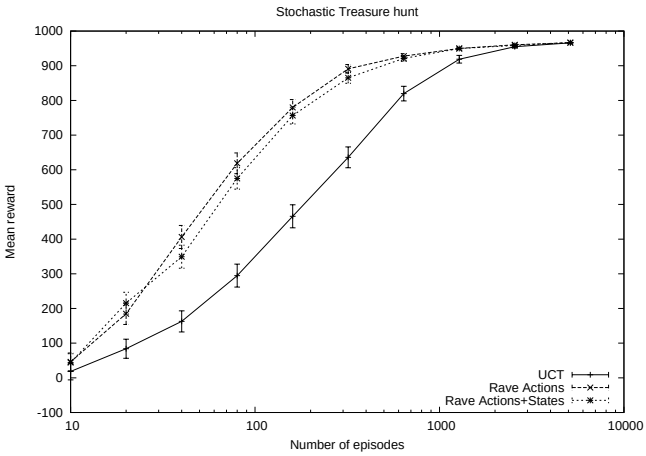
Figure 2(b) (medium and bottom) reports on the results in the probabilistic setting (respectively $\epsilon = .5$ and 1), where the optimal action $\pi(s)$ now depends



(a) Deterministic case, no trap.



(b) Stochastic case with $\epsilon = 0.5$, no trap.



(c) Stochastic case with $\epsilon = 1$, no trap.

Figure 2: Treasure hunt with 15×15 arena, without hole (top: deterministic transitions; middle and bottom: probabilistic transitions with respectively $\epsilon =$ .5 and 1).

on s.
In the probabilistic cases, both cRAVE$_{action,state}$ and cRAVE$_{action}$ clearly improve on UCT. Unexpectedly, cRAVE$_{action}$ outperforms cRAVE$_{action,state}$, all the more so as the noise is moderate. The proposed interpretation for this finding goes as follows: on the one hand, the estimate variance is lower when the state is not taken into account; on the other hand, the optimal decision only slightly depends on state s; overall, cRAVE$_{action}$ thus enforces a faster convergence of the estimate while its bias remains moderate. This interpretation is confirmed as the gap between cRAVE$_{action,state}$ and cRAVE$_{action}$ decreases with the noise amplitude ϵ.

The results obtained for the treasure hunt with a hole are reported in Figs. 3(a), 3(b) and 3(c). Clearly, the optimal move here depends on the current state, even in the deterministic transition setting. As expected, cRAVE$_{action,state}$ significantly improves on cRAVE$_{action}$ in all deterministic and probabilistic transition settings with the hole, although the gap decreases with the noise amplitude increasing. Further, both cRAVE$_{action}$ and cRAVE$_{action,state}$ improve on the baseline UCT.

4.3 Energy Management Problem

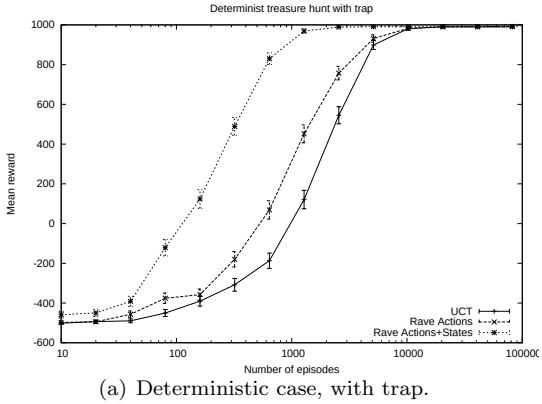
This real-world problem describes a power plant involving S stocks of energy (e.g. hydro-electric stocks); the time horizon is T. In each time step, the possible action is to produce a (continuous) quantity of electricity using any of the S stocks. The instant reward depends on the instant energy demand, a random variable. If the produced energy is less than the demand, the instant reward is negative (as the only management option is to buy extra energy and incur some pollution from the thermal power stations). The transition model boils down to decrementing the stock from the produced energy[1]. Historically this real-world problem is the applicative motivation of [10]'s and [3]'s seminal works on decomposition by dynamic programming.

Overall, the energy demand is supplied with i) the energy produced from the hydro-electric stocks; ii) if needed, the energy produced from the thermal power stations. In the latter case, an additional super-linear cost is incurred.

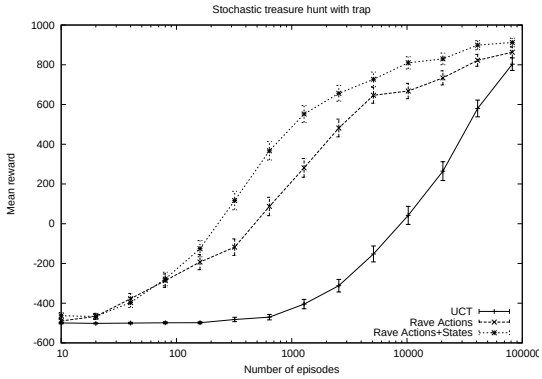
An optimal strategy must thus enforce a nearly constant thermal power production. The actual optimal strategy must however account for extra constraints and upper-bounds on the instant energy production, and the uncertainties on inflows. More precisely, there are random inflows for each stock, at each time step, where each inflow is a real random variable; all inflows are independent and identically distributed, following a uniform distribution over $[0, 1]$. The demand is fixed and known in advance; it depends on the time steps, to account for the seasonality of the consumption.

MCTS was investigated to find an optimal energy management policy within this setting, motivated by the fact that the underlying model of the power plants

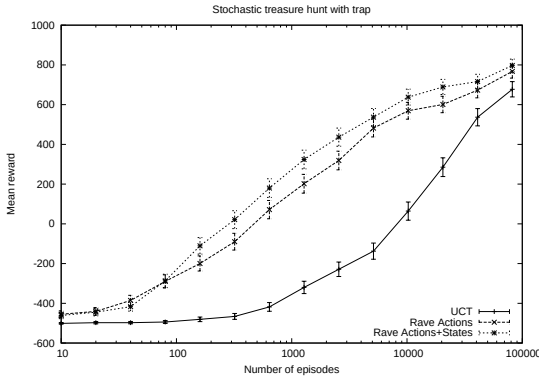
[1] Benchmark data have been gathered by the Iomca project `http://www.lri.fr/~teytaud/iomca.html`; these are available on demand to the last author.



(a) Deterministic case, with trap.



(b) Stochastic with $\epsilon = 0.5$, with trap.



(c) Stochastic with $\epsilon = 1.$, with trap.

Figure 3: Treasure hunt with 5×5 arena, with hole (top: deterministic transitions; middle and bottom: probabilistic transitions with respectively $\epsilon = .5$ and 1).

is non-linear and non-deterministic[2].

The experimental setting considers $S = 6$ stocks and $T = 12$ time horizon. The problem-dependent constants α_{action} and α_{state} are set to 10 (by consistency with the former treasure hunt problem, considering the range and dimension of action and state spaces, and to enforce the discrimination between different states and actions).

Figure 4 comparatively displays the results obtained by UCT, RAVE cRAVE$_{action}$ and cRAVE$_{action,state}$ on this problem. Interestingly, UCT is dominated by all other variants, including the RAVE variant devised to deal with discrete action spaces. Furthermore, cRAVE$_{action,state}$ significantly outperforms cRAVE$_{action}$; this finding was expected as two pairs (s, a) and (s', a') can only be considered similar if similar actions a and a' are applied on similar stock positions s and s'. For instance, the decision of using a minimal amount of water in order to use it later on, makes sense *if and only if* the stock positions are low, which is described through the current state. Overall, the merits of the cRAVE heuristics are fully empirically demonstrated on this simplified energy management problem.

5 Conclusion

The contribution proposed in this paper concerns the extension of the Rapid Action Value Estimate heuristics, originally proposed to prevent misleading exploration in large action spaces. RAVE has been extended to continuous action spaces (cRAVE$_{action}$, Eq. 6) using a Gaussian convolution; this approach was itself extended to the case of a continuous action and state spaces (cRAVE$_{action,state}$, Eq. 7). While these extensions can be easily plugged on the top of an UCT/RAVE algorithm, they only involve two additional hyper-parameters. The experimental validation of the approach on an artificial and a real-world problems fully demonstrates its potentialities, and its robustness w.r.t. some changes to the hyper-parameters.

A primary perspective for further work is to apply cRAVE in discrete domains where some distance/dissimilarity function can be defined using expert priors, e.g. classical game test beds like Go [9], Hex [1] or Havannah [14].

A longer-term perspective concerns the coupling of cRAVE$_{action,state}$ with the progressive widening (PW) heuristics. As already mentioned, PW introduces a new action in each state node from time to time, when the number of times this state has been visited reaches a given threshold. An interesting possibility would be to use $Q_{RAVE,a}$ and $Q_{RAVE,a,s}$ as value expectation, and select the *continuous* action a^* maximizing e.g. $Q_{RAVE,a,s}(s, a)$ over the whole action space $\mathcal{A}$.

[2]The linear deterministic case and the associated approaches are beyond the scope of the present paper.

Acknowledgements

This work was partially supported by the Pascal Network of Excellence. This work has been supported by French National Resear ch Agency (ANR) through COSINUS program (project EXPLO-RA nANR-08-COSI-004). It was also supported by the FP7 program under the Mash project (grant 247022). The last author is grateful to National Science Council (Taiwan) for grant NSC97-2221-E-024-011-MY2 and NSC 99-2923-E-024-003-MY3.

References

[1] B. Arneson, R. Hayward, and P. Henderson. Mohex wins hex tournament. *ICGA journal*, pages 114–116, 2009.

[2] P. Auer, N. Cesa-Bianchi, and P. Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine Learning*, 47(2/3):235–256, 2002.

[3] R. Bellman. *Dynamic Programming*. Princeton Univ. Press, 1957.

[4] 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.

[5] R. Coulom. Computing elo ratings of move patterns in the game of go. In *Computer Games Workshop, Amsterdam, The Netherlands*, 2007.

[6] F. De Mesmay, A. Rimmel, Y. Voronenko, and M. Püschel. Bandit-based optimization on graphs with application to library performance tuning. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 729–736. ACM, 2009.

[7] 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.

[8] L. Kocsis and C. Szepesvàri. Bandit based Monte-Carlo planning. In *15th European Conference on Machine Learning (ECML)*, pages 282–293, 2006.

[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] P. Massé. *Les Réserves et la Régulation de l'Avenir dans la vie Economique*. Herman, 1946.

[11] H. Nakhost and M. Müller. Monte-carlo exploration for deterministic planning. In C. Boutilier, editor, *IJCAI*, pages 1766–1771, 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] R. Sutton and A. G. Barto. *Reinforcement learning*. MIT Press, 1998.

[14] F. Teytaud and O. Teytaud. Creating an Upper-Confidence-Tree program for Havannah. In *ACG 12*, Pamplona Spain, 2009.

[15] B. Tuffin. On the use of low discrepancy sequences in monte-carlo methods. Technical Report Technical Report 1060, I.R.I.S.A., 1996.

[16] Y. Wang, J.-Y. Audibert, and R. Munos. Algorithms for infinitely many-armed bandits. In *Advances in Neural Information Processing Systems*, volume 21, 2008.

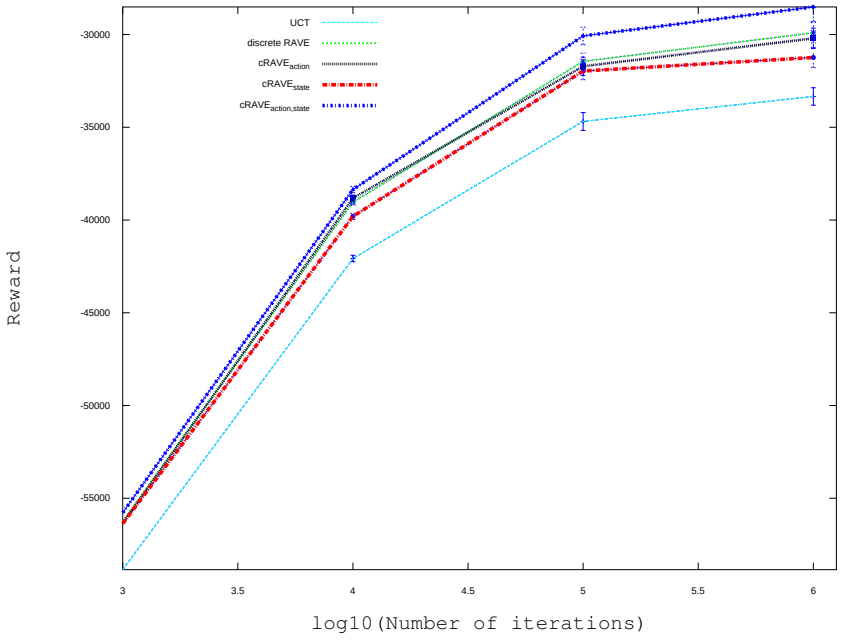


Figure 4: Comparative performances of UCT, cRAVE$_{action}$ and cRAVE$_{action,state}$ on the energy management problem, versus the computational budget (number of simulations). The upper the better.