Open Loop Execution of Tree-Search Algorithms

Erwan Lecarpentier[1,2], **Guillaume Infantes**[1], **Charles Lesire**[1], **Emmanuel Rachelson**[2]
[1] ONERA - The French Aerospace Lab, Toulouse, France
[2] ISAE - SUPAERO, University of Toulouse, France
erwan.lecarpentier@isae.fr, guillaume.infantes@onera.fr,
charles.lesire@onera.fr, emmanuel.rachelson@isae.fr

Abstract

In the context of tree-search stochastic planning algorithms where a generative model is available, we consider on-line planning algorithms building trees in order to recommend an action. We investigate the question of avoiding re-planning in subsequent decision steps by directly using sub-trees as action recommender. Firstly, we propose a method for open loop control via a new algorithm taking the decision of re-planning or not at each time step based on an analysis of the statistics of the sub-tree. Secondly, we show that the probability of selecting a suboptimal action at any depth of the tree can be upper bounded and converges towards zero. Moreover, this upper bound decays in a logarithmic way between subsequent depths. This leads to a distinction between node-wise optimality and state-wise optimality. Finally, we empirically demonstrate that our method achieves a compromise between loss of performance and computational gain.

1 Introduction

Tree-search based algorithms recently encountered a real success at solving sequential, highly combinatorial problems such as the challenging game of Go [Enzenberger *et al.*, 2010; Silver *et al.*, 2016]. Such algorithms use a generative model of the environment to simulate episodes starting from the current state of the agent [Sutton, 1991; Sutton and Barto, 1998]. This allows the exploration of reachable states and actions and results in the construction of an (unbalanced) scenario tree, that aims at identifying promising branches with a limited computational budget. When the computational budget is exhausted, the recommended action at the root node is applied and a new tree is built in the resulting state. This results overall in a closed loop control process.

We are interested in stochastic problems with large state spaces (e.g. continuous) with a short decision time (budget). In this setting, open loop planning algorithms have proven to be successful [Bubeck and Munos, 2010] and even to outperform [Weinstein and Littman, 2012] the standard approaches that consider closed loop policy trees such as UCT [Kocsis and Szepesvári, 2006]. They seek for optimal sequences of actions (plans) rather than optimal policies despite the sub-optimal nature of a plan in stochastic environments. Indeed, computing the latter prevents feed-back on the explored states but allows to break the complexity of the state space exploration. Given a tree computed by an open loop planning algorithm, we propose to keep the sub-tree reached by the application of the recommended action and to directly use it as the main tree for the subsequent time step, without re-planning. What motivates this approach is sparing the computational cost of tree building for subsequent time steps, hence reducing the number of calls to the simulator. The interest of this can be seen in two ways. On one hand it is a way of reducing energy consumption for systems with low computational resources [Wilson *et al.*, 2014; 2016]. On the other hand, the saved computational time can be re-invested into other tasks. Particularly, this approach is adapted for low level control (i.e. high frequency) where sub-sequent tree developments is cumbersome. In this framework, Perez *et al.* [2012a] and Heusner [2011] considered keeping the tree in deterministic environments but observed a negative impact as the sub-trees were systematically kept without analysis. Moreover, they lose the aforementioned computational gain by refining the sub-trees.

In this paper, we study the impact of using the subsequent sub-trees as main trees for the next action steps without further re-planning. We claim that in lowly-stochastic environments, the reached performance is comparable to algorithms systematically discarding the tree. Our contribution is threefold. (1) We introduce a new algorithm called OLTA (Section 3), performing a systematic analysis of the sub-tree and taking the decision of re-planning or not at each time step. (2) We upper bound the probability of selecting a suboptimal action within a sub-tree, the sense of optimality being defined in an open loop fashion (Section 4). Additionally, we show that this upper bound decays logarithmically with the sub-tree depth. (3) We show in our experiments the benefit of applying such a method both in terms of performance and computational cost saving (Section 5).

2 Background

2.1 Markov Decision Process

We model the planning problem as a Markov Decision Process (MDP) where an agent sequentially takes actions with the general goal of maximizing the cumulative return fed

back by the environment [Puterman, 2014]. We refer to the state space as S and the action space as A. We suppose the number of actions to be finite with $K = |A|$, thus we write $A = \{a_i\}_{i=1}^{K}$. We also consider that the available actions are independent of the state the agent lies in. The state transition function is stochastic and we note $P(s'|s,a)$ the probability of reaching state s' after taking action a in state s. The reward model is denoted by $r(s,a,s')$ and refers to the scalar reward received while performing the transition (s,a,s'). We assume that this reward function is deterministic. Finally, we suppose the horizon of the MDP is infinite and we note $\gamma \in [0,1)$ the discount factor which represents the importance of the subsequent collected rewards.

2.2 Tree Representation

When a generative model of the MDP is available, it becomes possible to use it within planning algorithms. Tree-search algorithms use this model in order to build a tree of what may possibly occur in the current situation of the agent [Sutton, 1991; Sutton and Barto, 1998; Silver *et al.*, 2008]. In the stochastic setting with potentially infinitely many states, we use a tree structure similar to the one used by Bubeck and Munos [2010]. The tree built at each time step consists in a look-ahead search of the possible outcomes while following some action plan starting from the current state of the agent $s_0 \in S$. Thus, the root node of the tree is labelled by the unique state s_0. The edges correspond to the K available actions, K being the branching factor of the tree. The tree itself conforms to an ensemble of action sequences, or plans, originating from its root node.

We emphasize the fact that this tree structure implies that we search for a state-independent optimal sequence of actions (open loop plan) which is in general sub-optimal compared to a state-dependent policy search. The THTS family of algorithms in particular [Keller and Helmert, 2013] defines trees with chance and decision nodes while our structure does not apply an equality operator on the sampled states. Following Bubeck and Munos; Weinstein and Littman [2010; 2012], we argue that closed-loop application of the first action in optimal open loop plans, although theoretically sub-optimal, can be competitive with these methods in practice, while being more sample-efficient.

Since the transition model is stochastic, the non-root nodes are not labelled by a unique state. Instead, every such node is associated to a state distribution resulting from the application of the action plan leading to the considered node and starting from s_0. During the exploration, we consider saving all sampled states at each non-root node. A comprehensive illustration of such a tree can be found in Figure 1. This approach extends straightforwardly to Partially Observable Markov Decision Processes (POMDP) [Silver and Veness, 2010].

Given a tree-search, open loop planning algorithm, we call $\mathcal{T}_d$ the **tree at depth** $d \in \mathbb{N}$, that is the sub-tree resulting from the application of the d first recommended actions. Hence $\mathcal{T}_0$ denotes the whole tree, $\mathcal{T}_1$ the tree starting from the node reached by the application of the first recommended action and so on.

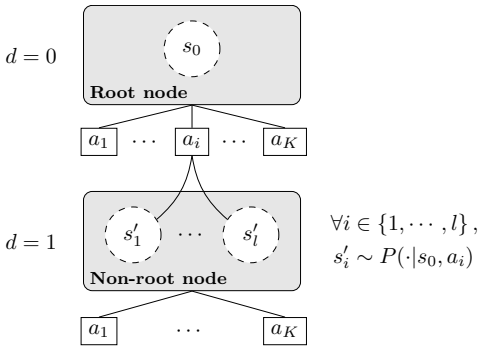


Figure 1: General representation of a tree, where $l \in \mathbb{N}$ is the number of times the sub-tree reached by action a_i has been developed. Two nodes are represented in this tree with their respective depths on the left.

2.3 Open Loop UCT

For the sake of clarity and in order to clearly separate the tree building properties from the open loop execution presented in the next section, we define an open loop planning algorithm utilizing the presented tree structure that we call Open Loop UCT (OLUCT). The difference between UCT and OLUCT is that OLUCT is not provided with an equality operator over states. Within the THTS terminology, this means that decision and chance nodes do not correspond to a single state but to the state distribution reachable by the action plan leading to the node. Hence decision and chance nodes are associated to the state distribution which makes OLUCT an open loop planning algorithm. The fundamental consequence is that an action value within our tree is computed w.r.t. the parent node's state distribution rather than a single state.

Apart from this, OLUCT uses the same exploration procedure as UCT. Within a node, we note $\overline{X}_{i,u}$ the estimated expected return of action i after u samples of this action. $T_i(t)$ is the number of trials of action i up to time t of the OLUCT procedure. An Upper Confidence Bound (UCB) strategy [Auer *et al.*, 2002] is applied at each node where each action is seen as an arm of a bandit problem. The tree policy selects the action I_t with the highest UCB:

$$I_t = \underset{i \in \{1,\cdots,K\}}{\arg\max} \left\{ \overline{X}_{i,T_i(t-1)} + c_{t-1,T_i(t-1)} \right\},$$

where $c_{t,u} = 2C_p\sqrt{\frac{ln(t)}{u}}$ is an exploration term ensuring that all actions will be sampled infinitely often.The C_p parameter drives the exploration-exploitation trade-off. The OLUCT tree building procedure is detailed in Algorithm 1.

3 OLTA (Open Loop Tree-search Algorithm)

3.1 Description

In order to control the execution of open loop plans, we propose a new algorithm called OLTA (Algorithm 2). It relies on a generic open loop planning algorithm to generate a tree, rooting from the current state. For the next execution time step, it decides either to use the sub-tree reached by the recommended action or to trigger a re-planning by building a new tree. If no re-planning is triggered, then the recommended action of the sub-tree is applied without using the

Algorithm 1: OLUCT tree building procedure

Function createTree(state s):
Parameters: budget n; default policy $\pi_{default}$.
Create root node $\nu_{root}(s)$
for $t \in \{1, \cdots, n\}$ **do**
  ν_{leaf} = Select(ν_{root}); // Select a leaf node w.r.t. the UCT strategy and sample a new state for each encountered node.
  Expand(ν_{leaf}); // Expand the node if not terminal using the generative model.
  Δ = Evaluate($\nu_{leaf}, \pi_{default}$); // Simulate a roll-out using $\pi_{default}$, starting from the last sampled state in ν_{leaf}.
  Backup(ν_{leaf}, Δ); // Back-propagate the sampled return.
return $\mathcal{T}(\nu_{root})$

Algorithm 2: OLTA algorithm

Function OLTA:
Parameters: initial state s_0; tree building procedure createTree; re-planning criterion decisionCriterion.
$s = s_0$;
$\mathcal{T}$ = createTree(s);
while *s is not terminal* **do**
  if *decisionCriterion*($s, \mathcal{T}$) **then**
    a = recommendedAction($\mathcal{T}$); // Get the first recommended action.
  else
    $\mathcal{T}$ = createTree(s); // Create a new tree from the current state.
    a = recommendedAction($\mathcal{T}$); // Get the first recommended action.
  $\mathcal{T}$ = subTree($\mathcal{T}, a$); // Move the tree to the first sub-tree resulting from the application of a.
  s := realWorldTransitionFunction(s, a);

additional information of the new state observed after the transition. This results in an open loop control process and spares the cost of developing a new tree starting at this state. The intuition behind OLTA is that several consecutive recommended actions in an optimal branch of the tree can be reliable, despite the randomness of the environment. A major example of such a case is low-level control, where consecutive sampled states are close to each other.

In this paper, for its performance and simplicity, we chose to implement OLUCT as the open loop planning algorithm utilized by OLTA. However, any other algorithm generating trees as described in Section 2.2 could be used in the same way (e.g. OLOP [Bubeck and Munos, 2010], or HOLOP [Weinstein and Littman, 2012]).

One important feature of OLTA is the so-called "*decisionCriterion*", based on which the agent decides to either use the first sub-tree following the recommended action, or to rebuild a new tree from the current state. The decision is based on a comparison with the characteristics of the resulting subtree and the current state of the agent. In the next section, we discuss different decision criteria, leading to the consideration of a family of different algorithms.

3.2 Decision Criterion

The simplest implementation of the decision criterion is to keep the sub-tree only if its root node is fully expanded. This means that each action has been sampled at least once. We call the resulting algorithm **Plain OLTA**. It naively trusts the value estimates of the sub-tree, thus applies the whole plan of recommended actions at each depth until it reaches a partially expanded node. Therefore, Plain OLTA is expected to perform better in deterministic environments. In stochastic cases however, those estimates may be biased because of the different sources of uncertainty within the MDP (reward function, state transition function and action selection). For this reason, we seek more robust criteria to base the decision on.

A natural way to decide whether to keep the sub-tree or not is to track if the recommended action is optimal w.r.t. the new state s of the agent. Here we make an important distinction between a **state-wise optimal action** and a **node-wise optimal action**. The first one is the action recommended by the optimal policy in a specific state. We note it $a^* = \arg\max_{a \in A} Q^*(s, a)$, with $Q^* : S \times A \to \mathbb{R}$ the optimal state-action value function. In order to define the second one, we introduce S_d, the state random variable at the root node of $\mathcal{T}_d$. Its distribution results from the application of the d first recommended actions starting from s_0, so $S_d \sim P(\cdot|s_0, a_0, \cdots, a_{d-1}) \equiv P_{S_d}(\cdot)$. The node-wise optimal action maximizes the expected return given the state *distribution* of the node. We note it $a^*_d = \arg\max_{a \in A} Q^*_d(a)$ where $Q^*_d : A \to \mathbb{R}$ is the optimal action value function w.r.t. the state distribution at the root node of $\mathcal{T}_d$, that is $Q^*_d(a) = \mathbb{E}_{s \sim P_{S_d}}(Q^*(s, a))$. Following Bellemare *et al.* [2017], a distributional Bellman equation can be expressed in terms of three sources of randomness that are: $R : S \times A \to \mathbb{R}$ the stochastic reward function; $X : S \times A \to \mathbb{R}$ the random return; and P^π the transition operator with $P^\pi X(s, a) \overset{D}{:=} X(S', A')$, $S' \sim P(\cdot|s, a)$ and $A' \sim \pi(\cdot|S')$. Mathematically, we have the following distributional Bellman equations:

$$\begin{cases} Q^\pi(s, a) = \mathbb{E}_\pi(X(s, a)) \\ Q^\pi_d(a) = \mathbb{E}_{P_{S_d}}(Q^\pi(s, a)) \end{cases},$$

with $X(s, a) \sim R(s, a) + \gamma P^\pi X(s, a)$ and $S_d \sim P_{S_d}(\cdot)$. Unfortunately, at the root node of $\mathcal{T}_d$ for $d > 0$, open loop tree-search algorithms do not estimate Q^* but Q^*_d. The bias introduced by the state distribution implies that in the general case we have no guarantee that $a^* = a^*_d$. The risk is that the set Ω_{S_d} of possible realizations of S_d can include states where a^* is sub-optimal, in which case the resulting return evaluations would weight in favour of a different action than a^*. In other words — introducing the notion of domination domain for an action a as $\mathcal{D}_a = \{s \in S | \pi^*(s) = a\} \subset S$ — if Ω_{S_d} is not included in $\mathcal{D}_{a^*}$, then the risk of the recommended action to be state-wise sub-optimal is increased. Conversely, if Ω_{S_d} is included in the domination domain of a^*, then the optimal

action will be selected given that the budget is "big enough" w.r.t. the chosen tree-search algorithm's performance. Consequently, one should base the decision criterion on the analysis of P_{S_d} and the action domination domains. To compute these domains, Rachelson and Lagoudakis [2010] use the properties of Lipschitz-MDPs. Although the following discussion is inspired by this work, the consideration of Lipschitz-MDPs is out of the scope of this paper. We discuss below the construction of decision criteria that will be illustrated in Section 5.

Current state analysis & POMDP setting. The current state s of the agent can be compared to the empirical state distribution $\overline{P_{S_d}}$ at the root node of the sub-tree. If $P_{S_d}(s)$ is large, then the value estimators are related to the locality of the state space the agent lies in. If not, then the node-wise optimal action may not be state-wise optimal. This consideration supposes to identify a state-metric for which two close states have a high chance to be in the same action domination domain. Alternatively, in the case of a POMDP, a belief distribution on the current state is available instead of the current state itself [Kaelbling *et al.*, 1998]. In such a case, a direct comparison between this distribution and $\overline{P_{S_d}}$ can be performed (e.g. with a Wasserstein metric). Note that making use of the current state of the agent makes the algorithm closed-loop, by definition. We use the terminology "open-loop" in order to distinguish OLTA from classical closed-loop Tree Search algorithms that systematically re-plan, rooting from the current state (e.g. OLOP [Bubeck and Munos, 2010], performs closed-loop execution).

State distribution analysis. The dispersion and multi-modality of $\overline{P_{S_d}}$ could motivate not to re-use a sub-tree. A high dispersion involves the possibility that $\overline{\Omega_{S_d}}$ does not belong to a single action domination domain and a re-planning should be triggered. The same consideration applies in terms of multi-modality. Conversely, a narrow, mono-modal, state distribution is a good hint for Ω_{S_d} to be comprised into a single action domination domain.

Return distribution analysis. A widespread or a multi-modal return distribution for the recommended action in a node may indicate a strong dependency on the region of the state space we lie in. If Ω_{S_d} covers different action domination domains, each of these domains may contribute a different return distribution to the node's return estimates, thus inducing a high variance on this distribution or even a multi-modality. In this case, it could be beneficial to trigger the re-planning. Alternatively, even after re-planning, widespread or multi-modal return distributions can naturally arise as a result of the MDP's reward and transition models.

We do not provide a unique generic method to base the decision criterion on. Indeed, we believe that it is a strongly problem-dependent issue and that efficient heuristics can be built accordingly. However, the analysis of the state and return distributions constitute promising indicators and we exemplify their use in the experiments of the last section.

4 Theoretical Analysis

In this section, we demonstrate that the algorithm asymptotically provides node-wise optimal actions for any sub-tree $\mathcal{T}_d$ of depth d. We first derive an upper bound on the failure probability that converges towards zero when the initial budget n of the algorithm goes to infinity. Then, we characterize the loss of performance guarantees between subsequent depths and show a logarithmic decay of the upper bound. The demonstration unfolds as follows: first we write a lower bound for the number of trials of the actions at the root of $\mathcal{T}_d$ in Lemma 1; then we write an upper bound on the failure probability given a known budget at depth d in Lemma 2; finally we derive a recursive relation between the upper bounds of subsequent trees that leads to our result in Theorem 1. Proofs are omitted due to lack of space[1].

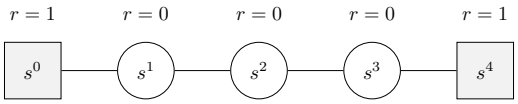


Figure 2: 1D track environment. On top of each cell representing a state is the immediate reward of the transition to this state.

We note $b(d) \in \mathbb{N}$ the **budget** used to develop $\mathcal{T}_d$ i.e. the number of times the d first recommended actions have been selected by the tree policy. We note $T^d_{i,t}$ the number of times the i^{th} action at the root node of $\mathcal{T}_d$ has been selected by the OLUCT tree policy after t expansions of $\mathcal{T}_d$. Similarly, $\overline{X}^d_{i,T^d_{i,t}} \equiv \overline{X}^d_{i,t}$ denotes the estimate of the return of the i^{th} action at depth d after t expansions of the sub-tree $\mathcal{T}_d$. We write I^d_t the index of the action chosen by the tree policy at depth d after t expansions of $\mathcal{T}_d$. We have:

$$I^d_t = \underset{i\in\{1,\cdots,K\}}{\arg\max} \left\{ \overline{X}^d_{i,t-1} + c_{t-1,T^d_{i,t-1}} \right\}.$$

The recommended action at depth d given a budget $b(d)$ is $\widehat{I}^d = \arg\max_{i\in\{1,\cdots,K\}} \overline{X}^d_{i,b(d)}$. Following Kocsis and Szepesvári [2006], we assume that the empirical estimates $\overline{X}^d_{i,t}$ converge and write $X^d_{i,t} = \mathbb{E}\{\overline{X}^d_{i,t}\}$ and $X^d_i = \lim_{t\to\infty} X^d_{i,t}$. Then, we define for $i \in \{1,\cdots,K\} \setminus i^*_d$, $\Delta^d_i = X^d_{i^*_d} - X^d_i$ where we note i^*_d the index of the node-wise optimal action at the root node of $\mathcal{T}_d$. We make the assumption that only one action is optimal in a given node. The minimum return difference between a suboptimal action and the optimal one at depth d is $\delta^d = \min_{i\in\{1,\cdots,K\}\setminus i^*_d}(\Delta^d_i)$.

Lemma 1. ***Lower bound for the number of trials.*** *For any sub-tree $\mathcal{T}_d$ developed with a budget $b(d) > K$, there exist a constant $\rho \geq 0$ such that $T^d_{i,b(d)} \geq \lceil \rho \ln(b(d)) \rceil$ for all $i \in \{1,\cdots,K\}$. Furthermore, we have the following sequence of lower bounds for the budget with $\lceil\cdot\rceil$ the ceiling function:*

$$\begin{cases} b(d=0) = n \\ b(d) \geq \lceil \rho \ln(b(d-1)) \rceil \end{cases}.$$

Lemma 2. ***Upper bound on the failure probability at depth d given the budget*** $b(d)$**.** *For any sub-tree $\mathcal{T}_d$ developed with a budget $b(d) > K$ we have the following upper bound on the failure probability, conditioned by the budget $b(d)$:*

$$P(\widehat{I}^d \neq i^*_d | b(d)) \leq b(d)^{-\frac{\rho}{2}(\delta^d)^2}.$$

[1] A longer version of the paper including the proofs can be found at https://arxiv.org/abs/1805.01367

Theorem 1. ***Upper bound on the failure probability at depth*** *d. For an initial budget of n and for any sub-tree $\mathcal{T}_d$ developed with a budget $b(d) > K$, we have the following recursive relation for the upper bound on the failure probability, conditioned by the initial budget n:*

$$P(\widehat{I}^d \neq i_d^*|n) \leq \lceil \rho \ln(b(d-1)) \rceil^{-\frac{\rho}{2}(\delta^d)^2}.$$

Additionally, for any depth $d \geq 1$ given the initial budget n:

$$\begin{cases} P(\widehat{I}^d \neq i_d^*|n) \leq f^d(n)^{-\frac{\rho}{2}(\delta^d)^2} \\ f : t \mapsto \lceil \rho \ln(t) \rceil \end{cases}.$$

Where $f^d = f \circ f^{d-1}$ with $f^1 = f$ and $d > 0$.

This result shows a logarithmic decay between the upper bounds on the failure probability of two subsequent trees. Asymptotically, at any depth, this upper bound converges towards zero. This result highlights the fact that the deeper the sub-tree is, the less one can rely on the recommended action at the root node. However, we should note that these upper bounds are derived without making further hypotheses on the MDP and express a worst-case value. We show in the next section that equal performances to OLUCT can be reached with a smaller computational budget and number of calls to the generative model.

5 Empirical Analysis

We compared OLUCT with OLTA on a discrete 1D track environment[2] and a continuous Physical Travelling Salesman Problem[3] (PTSP) [Perez *et al.*, 2012b]. We implemented five decision criteria, leading to five variations of OLTA.

5.1 Heuristic decision criteria

A relevant decision criterion w.r.t. the treated problem allows OLTA to discard a sub-tree when its first recommended action may not be state-wise optimal given the current state of the agent. We implemented five different tests to base this decision on, and evaluated them independently, which led to the following variations of OLTA.

Plain OLTA. The simplest decision criterion that discards a sub-tree only if its root-node is not fully expanded.

State Distribution Modality (SDM-OLTA). Test whether the empirical state distribution is multi-modal or not. If yes, discard the tree if the current state of the agent does not belong to a majority mode. We define a majority mode by a mode comprising more than $\tau_{SDM}\%$ of the sampled states.

State Distribution Variance (SDV-OLTA). Test whether the empirical state distribution variance is above a certain threshold τ_{SDV}. Discard the tree if it is the case. For multi-dimensional state spaces such as in the PTSP, the Variance-Mean-Ratio (VMR) is considered for the different orders of magnitude to be comparable.

State Distance to State Distribution (SDSD-OLTA). Compute the Mahalanobis distance [De Maesschalck *et al.*, 2000] of the current state from the empirical state distribution. Discard the tree if it is above a selected threshold τ_{SDSD}.

Return Distribution Variance (RDV-OLTA). Test whether the empirical return distribution variance is above a certain threshold τ_{RDV}. Discard the tree if it is the case.

A more selective decision criterion can easily be derived by combining the previously described decision criteria and discarding the tree if one of them recommends to do so.

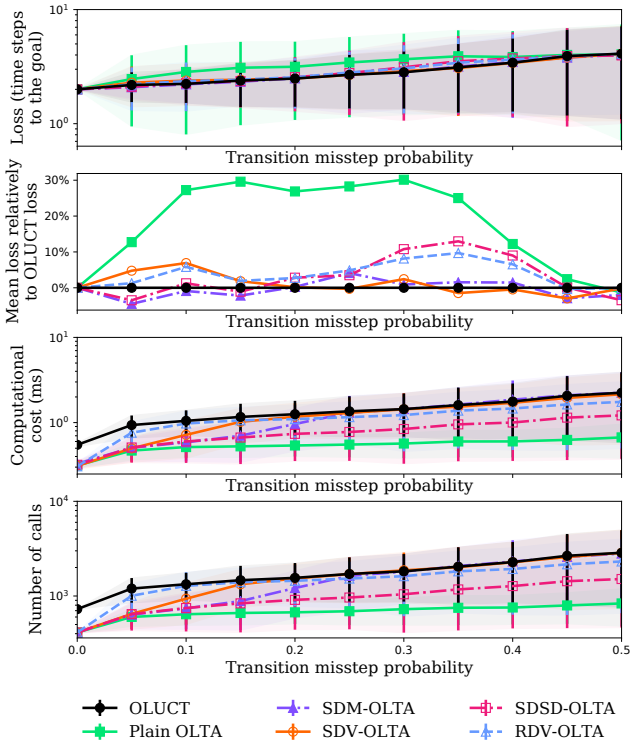
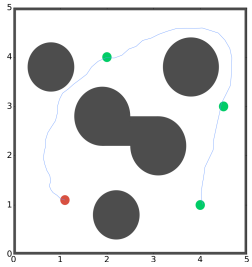


Figure 3: Comparison between OLUCT and OLTA on the discrete 1D track environment for varying values of q.

5.2 1D Track Environment

The 1D track environment (Figure 2), is a 1D discrete world where an agent can either go right or left. The initial state is the "middle" state $s_0 = s^2$. The reward is 0 everywhere except for the transition to the two terminal states s^0 and s^4 for which it is $+1$. The action space is $A = \{right, left\}$. We introduce a transition misstep probability $q \in [0, 1]$ which is the probability to end up in the opposite state after taking an action, for $i \in \{1, 2, 3\}$: $P(s^{i-1}|s^i, right) = q$ and $P(s^{i+1}|s^i, right) = 1 - q$. The same applies for the $left$ action. If $q < 0.5$, the optimal policy $\pi_{optimal}$ is to go left at s^1, to act randomly at s^2 and to go right at s^3. The simulation settings are: $q \in \{0.0, 0.05, \cdots, 0.5\}$; $n = 20$ (budget); $\pi_{default} = \pi_{optimal}$; $H = 10$ (simulation horizon for $\pi_{default}$); $C_p = 0.7$; $\gamma = 0.9$. The decision criteria parameters were tuned to: $\tau_{SDM} = 80$; $\tau_{SDV} = 0.4$; $\tau_{SDSD} = 1$; $\tau_{RDV} = 0.9$. We generated 1000 episodes for each value of q and recorded 3 performance measures: loss (number of time steps to termination); computational cost (measured computation time); and number of calls to the generative model. We

Code available at:
[2]https://github.com/erwanlecarpentier/1dtrack.git
[3]https://github.com/erwanlecarpentier/flatland.git



Trajectory derived by an OLUCT algorithm in our PTSP setting. The starting point is displayed in red, the waypoints in green and the walls in grey.

Figure 4: PTSP illustration

display two different graphs of the loss, the second one highlights the relative performance between OLTA and OLUCT.

The motivation behind the use of such a benchmark is to test open loop control in a highly stochastic environment where feedback of the current state is highly informative about the optimal action. In case of misstep for the first action, OLTA has to guess that a re-planning should be triggered while OLUCT does it systematically. As seen on Figure 3, the non-plain OLTA and OLUCT achieved a very comparable loss. Plain-OLTA had a weaker performance due to its systematic re-use of the sub-trees. Notice that some variations of OLTA such as SDV-OLTA achieved a better mean loss than OLUCT for some values of q. Due to the high variance, this observation cannot lead to the conclusion that OLTA can outperform OLUCT. However, this emphasizes the fact that the performance are very similar. In terms of both computational cost and number of calls to the generative model, OLTA widely outperforms OLUCT. As q increases, this computational gain vanishes and catches up with OLUCT for SDM-OLTA and SDV-OLTA. This accounts for the discriminative power of their decision criteria that discard more trees. RDV-OLTA and SDSD-OLTA kept a lower computational cost while reasonably matching the performance of OLUCT. Obviously, the computational cost of Plain-OLTA stays low. The apparent similarity between the number of calls to the generative model and the computational cost proves that computing our decision criteria is less expensive than re-planning.

5.3 Physical Travelling Salesman Problem

The PTSP is a continuous navigation problem in which an agent must reach all the waypoints within a maze (Figure 4). The state of the agent is $s = (x, y, \theta, v) \in \mathbb{R}^4$ i.e. the 2D position, orientation and velocity. The action space is $A = \{+d\theta, 0, -d\theta\}$ which consists of the increment, decrement or no-change of the orientation. The reward is $+1$ when a waypoint is reached for the first time, -1 for a wall crash and 0 otherwise. The simulation terminates when the agent reaches all the waypoints or a time limit. The walls cannot be crossed and the orientation is flipped when a crash occurs. We introduce a misstep probability $q \in [0, 1]$ which is the probability for another action to be undertaken instead of the current one. A Gaussian noise of standard deviation σ_{noise} is added to each component of the resulting state from a transition. The simulation settings are: $s_0 = (1.1, 1.1, 0, 0.1)$; $q \in \{0.0, 0.05, \cdots, 0.5\}$; $\sigma_{noise} = 0.02$; $n = 300$ (initial tree budget); $\pi_{default} = \pi_{go-straight}$ that applies no orientation variation; $H = 50$ (simulation horizon for $\pi_{default}$);

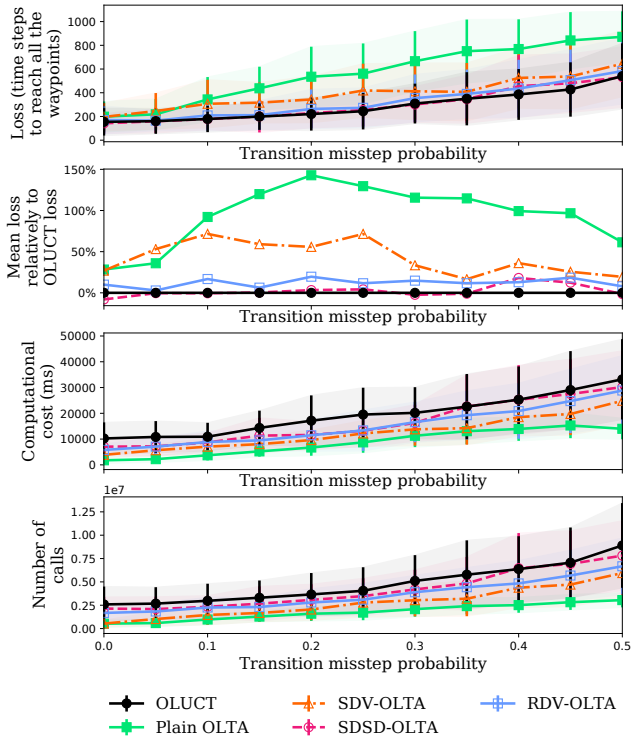


Figure 5: Comparison between OLUCT and OLTA on the continuous PTSP for varying values of q.

$C_p = 0.7$; $\gamma = 0.99$. The provided map is the one depicted in Figure 4 with three waypoints. The different decision criteria parameters were tuned to: $\tau_{SDV} = 0.02$; $\tau_{SDSD} = 1$; $\tau_{RDV} = 0.1$. We reserve the development of SDM-OLTA in the continuous case for future work. We generated 100 episodes for each transition misstep probability and recorded the same performance measures as in the 1D track case. The results are presented in Figure 5. OLUCT, SDSD-OLTA and RDV-OLTA achieved a comparable loss for every q, which shows that our method is applicable to larger scale problems than the 1D track environment. SDV-OLTA reached a lower level of performance. Plain OLTA still realized the highest loss since it is highly sensitive to the stochasticity of the environment. In terms of both computational cost and number of calls to the generative model, the same trade-off between performance and computational cost is observed. Plain OLTA and SDV-OLTA considerably lowered the number of calls at the cost of the performance while SDSD-OLTA and RDV-OLTA realized a better compromise. The number of calls to the generative model and the computational cost are quite similar, meaning that — even with the higher dimensionality of the PTSP compared to the 1D track — the cost incurred by the decision criteria computation is negligible in comparison to the one incurred by the re-planning procedure. Notice that SDV-OLTA achieved a good cost-performance trade-off in the 1D track environment while not in the PTSP relatively to the other algorithms. This is explained by the decision criteria's sensitivity to parameter tuning and by the problem-dependent relevance of such a criterion.

6 Conclusion

We introduced OLTA, a new class of tree-search algorithms performing open loop control by re-using subsequent sub-trees of a main tree built with the OLUCT algorithm. A decision criterion based on the analysis of the current sub-tree allows the agent to efficiently determine if the latter can be exploited. Practically, OLTA can achieve the same level of performance as OLUCT given that the decision criterion is well designed. Furthermore, the computational cost is strongly lowered by decreasing the number of calls to the generative model. This saving is the main interest of the approach and can be exploited in two ways: it decreases the energy consumption which is relevant for critical systems with low resources such as Unmanned Vehicles or Satellites; It allows a system to re-allocate the computational effort to other tasks rather than controlling the robot. We emphasize the fact that this method is generic and can be combined with any other tree-search algorithm than OLUCT. Open questions include building non problem-dependent decision criteria, e.g. by making more restrictive hypothesis on the considered class of MDPs, but also applying the method to other benchmarks and other open loop planners.

Acknowledgements

This research was supported by the Occitanie region, France.

References

[Auer *et al.*, 2002] Peter Auer, Nicolo Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine learning*, 47(2-3):235–256, 2002.

[Bellemare *et al.*, 2017] Marc G. Bellemare, Will Dabney, and Rémi Munos. A distributional perspective on reinforcement learning. *arXiv preprint arXiv:1707.06887*, 2017.

[Bubeck and Munos, 2010] Sébastien Bubeck and Rémi Munos. Open loop optimistic planning. In *COLT*, 2010.

[De Maesschalck *et al.*, 2000] Roy De Maesschalck, Delphine Jouan-Rimbaud, and Désiré L. Massart. The Mahalanobis distance. *Chemometrics and intelligent laboratory systems*, 50(1):1–18, 2000.

[Enzenberger *et al.*, 2010] Markus Enzenberger, Martin Muller, Broderick Arneson, and Richard Segal. Fuego - an open-source framework for board games and Go engine based on Monte Carlo tree search. *IEEE Transactions on Computational Intelligence and AI in Games*, 2(4):259–270, 2010.

[Heusner, 2011] Manuel Heusner. UCT for pac-man. Bachelor thesis, Univ. of Basel, 2011.

[Kaelbling *et al.*, 1998] Leslie Pack Kaelbling, Michael L. Littman, and Anthony R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial intelligence*, 101(1):99–134, 1998.

[Keller and Helmert, 2013] Thomas Keller and Malte Helmert. Trial-based heuristic tree search for finite horizon MDPs. In *ICAPS*, 2013.

[Kocsis and Szepesvári, 2006] Levente Kocsis and Csaba Szepesvári. Bandit based Monte Carlo planning. In *ECML*, volume 6, pages 282–293. Springer, 2006.

[Perez *et al.*, 2012a] Diego Perez, Philipp Rohlfshagen, and Simon M. Lucas. Monte Carlo tree search: Long-term versus short-term planning. In *Computational Intelligence and Games (CIG), 2012 IEEE Conference on*, pages 219–226. IEEE, 2012.

[Perez *et al.*, 2012b] Diego Perez, Philipp Rohlfshagen, and Simon M. Lucas. The physical travelling salesman problem: WCCI 2012 competition. In *Evolutionary Computation (CEC), 2012 IEEE Congress on*, pages 1–8. IEEE, 2012.

[Puterman, 2014] Martin L. Puterman. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.

[Rachelson and Lagoudakis, 2010] Emmanuel Rachelson and Michail G. Lagoudakis. On the locality of action domination in sequential decision making. In *ISAIM*, 2010.

[Silver and Veness, 2010] David Silver and Joel Veness. Monte Carlo planning in large POMDPs. In *Advances in neural information processing systems*, pages 2164–2172, 2010.

[Silver *et al.*, 2008] David Silver, Richard S. Sutton, and Martin Müller. Sample-based learning and search with permanent and transient memories. In *Proceedings of the 25th international conference on Machine Learning*, pages 968–975. ACM, 2008.

[Silver *et al.*, 2016] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

[Sutton and Barto, 1998] Richard S. Sutton and Andrew G. Barto. *Reinforcement learning: An introduction*. MIT press Cambridge, 1998.

[Sutton, 1991] Richard S. Sutton. Dyna, an integrated architecture for learning, planning, and reacting. *ACM SIGART Bulletin*, 2(4):160–163, 1991.

[Weinstein and Littman, 2012] Ari Weinstein and Michael L. Littman. Bandit-based planning and learning in continuous-action markov decision processes. In *ICAPS*, 2012.

[Wilson *et al.*, 2014] Mark A. Wilson, James McMahon, and David W. Aha. Bounded expectations for discrepancy detection in goal-driven autonomy. In *AI and Robotics: Papers from the AAAI Workshop*, 2014.

[Wilson *et al.*, 2016] Mark A. Wilson, James McMahon, Artur Wolek, David W. Aha, and Brian H. Houston. Toward goal reasoning for autonomous underwater vehicles: Responding to unexpected agents. In *Goal Reasoning: Papers from the IJCAI Workshop*, 2016.