Proceedings of Machine Learning Research 129:577–592, 2020 ACML 2020

Monte-Carlo Graph Search: the Value of Merging Similar States

Edouard Leurent EDOUARD.LEURENT@INRIA.FR
Odalric-Ambrym Maillard ODALRIC.MAILLARD@INRIA.FR
Inria Lille – Nord Europe, 40 avenue Halley, 59650 Villeneuve d'Ascq, France

Editors: Sinno Jialin Pan and Masashi Sugiyama

Abstract

We consider the problem of planning in a Markov Decision Process (MDP) with a generative model and limited computational budget. Despite the underlying MDP transitions having a graph structure, the popular Monte-Carlo Tree Search algorithms such as `UCT` rely on a tree structure to represent their value estimates. That is, they do not identify together two similar states reached via different trajectories and represented in separate branches of the tree. In this work, we propose a *graph-based* planning algorithm, which takes into account this state similarity. In our analysis, we provide a regret bound that depends on a novel problem-dependent measure of difficulty, which improves on the original tree-based bound in MDPs where the trajectories overlap, and recovers it otherwise. Then, we show that this methodology can be adapted to existing planning algorithms that deal with stochastic systems. Finally, numerical simulations illustrate the benefits of our approach.

Keywords: Online Planning, Tree-search, Reinforcement Learning.

1. Introduction

Monte Carlo tree search (MCTS) algorithms (Coulom, 2006) were a breakthrough for online decision-making in Markov decision processes (MDPs), that lead to key successes in the domain, including Computer Go (Silver et al., 2018). They enjoy two main benefits: first, they do not require the knowledge of the MDP parameters contrary to *e.g.* Dynamic Programming algorithms, but only the access to a *generative model* that allows to sample trajectories from the current state. Second, the theoretical performance bounds of MCTS algorithms are typically independent of the size of the state space. Instead, they depend on the maximum depth at which an optimal node in the search tree can be reached within the allowed *budget* of trajectory samples. This translates as an *effective branching factor* in the bounds, related to the notion of near-optimality dimension in multi-armed bandits.

Algorithms for planning with a generative model date back at least to the seminal work of Kearns et al. (2002) who proposed the `Sparse Sampling` algorithm using a tree structure to represent the value estimate and uniform sampling of trajectories. This strategy was further analysed more recently in (Feldman and Domshlak, 2014), where the `BRUE` algorithm provides an enhanced value estimation. Another family of algorithms rely on the principle of *Optimism in the Face of Uncertainty* (surveyed by Munos, 2014), inspired from the Multi-Armed Bandit problem. This principle was first used in the context of planning in the `CrazyStone` software (Coulom, 2006) for computer Go. It was later formalized with

the UCT algorithm (Kocsis and Szepesvári, 2006), but was shown by Coquelin and Munos (2007) to have a doubly-exponential complexity in the worst case. The Optimistic Planning for Deterministic Systems (OPD) algorithm introduced by Hren and Munos (2008) was the first to provide a polynomial regret bound, but was limited to systems with deterministic rewards and dynamics. It was then extended to the case of stochastic rewards (Bubeck and Munos, 2010; Leurent and Maillard, 2019) with deterministic transitions. Known stochastic transitions were handled by Busoniu and Munos (2012). For MDPs with stochastic and unknown transitions, polynomial sample complexities have been obtained for StOP (Szorenyi et al., 2014), TrailBlazer (Grill et al., 2016) and SmoothCruiser (Grill et al., 2019), but despite their theoretical merits these algorithms are intractable in practice: StOP requires the expensive storage of policies, while TrailBlazer and SmoothCruiser only terminate after a prohibitive amount of samples, even for very small MDPs. (Huang et al., 2017; Kaufmann and Koolen, 2017) proposed two algorithms for planning in a maxmin game with stochastic rewards in the leaves of a known game tree. The latter was recently extended by Jonsson et al. (2020) with the MDP-GapE algorithm for planning in unknown stochastic MDPs, which enjoys a gap-dependent sample complexity.

Despite its simplicity, the use of a tree structure comes with a limitation: MCTS algorithms *do not merge information across states*. That is, if a state s can be reached via two trajectories, it will be represented twice in the look-ahead tree. For instance, in Figure 1 (left), two paths lead to the same state represented in orange. MCTS algorithms do not merge the information of the two trajectories to update a shared estimate of the state value.

Related work The idea of merging information between branches of a search tree appears in (Silver et al., 2018), where the state values are approximated with a shared Neural Network. However, this network is merely updated between two planning instances and not during the planning procedure itself. Another work of interest is that of Hostetler et al. (2014), who propose to partition the state space S into a smaller set $\mathcal{X}$ of equivalence classes. By aggregating similar states within a class, they reduce the branching factor of the search tree from $|S||A|$ to $|\mathcal{X}||A|$, which substantially improves sample complexity as they illustrate empirically. However, this procedure requires providing a relevant state partition, only aggregates trajectories that traverse the same sequence of classes (*i.e.* local deformations), and comes with a (bounded) loss of optimality. The closest work to ours is that of Ballesteros et al. (2013), in the context of partially observable MDPs, who identify similar belief states and plan with a graph structure. This work focuses on empirically comparing various similarity measures on robotic tasks and does not provide any theoretical analysis of the effect of aggregation. This is precisely our goal and contribution here.

Contributions We introduce a planning algorithm named GBOP-D, a graph-based version of the tree-based OPD algorithm for deterministic systems. We analyse the benefits of this graph-based formulation in Section 4, and provide in Theorem 16 a regret guarantee. The corresponding regret bound features a novel problem-dependent difficulty measure that we introduce to capture the benefit of using a graph structure. We show that this measure can only improve over the performance of OPD, and provide an example where it does. We discuss in Section 5 an extension of our method to stochastic MDPs, called GBOP. Finally, Section 6 illustrates the benefits of GBOP in two numerical simulations.

2. Background

In a *Markov Decision Process* (MDP), an agent observes its current state s from a state space S and picks an action a from an action space A of size K, before transitioning to a next state s' drawn from a transition distribution $P(s' \mid s, a)$ and receiving a bounded reward $r \in [0, 1]$ drawn from a reward distribution $P(r \mid s, a)$. The goal of the agent is to maximise in expectation its cumulative discounted rewards $\sum_{t=0}^{\infty} \gamma^t r_t$, where $\gamma \in (0, 1)$ is a discount factor. This amounts to choosing at each step the action that maximises the state-action value function $Q(s, a) \overset{def}{=} \max_\pi \mathbb{E}_{\tau\sim\pi}\left[\sum_{t=0}^{\infty} \gamma^t r_t \;\middle|\; s_0 = s, a_0 = a\right]$ where π is a policy mapping states to actions and $\tau = (s_0, a_0, \dots)$ is a trajectory generated by following π. In *Monte-Carlo planning*, the underlying MDP is *unknown* to the agent which only has access to a *generative model* that provides samples s' of $P(s' \mid s, a)$ when queried. Then, under computational constraints known as *fixed-budget*, the agent is only allowed a limited budget n of queries to the generative model before recommending a good action a_n to take next. The quality of that action is assessed in terms of the *simple regret*

$$r_n \overset{def}{=} V(s) - Q(s, a_n), \text{ where } V(s) \overset{def}{=} \max_a Q(s, a). \tag{1}$$

3. Graph-Based Planning for Deterministic Systems

In this section, we introduce a simple yet highly effective variant of tree-based planning algorithms. We first consider the simple setting of MDPs with deterministic dynamics and rewards, and will denote $r(s, a)$ the unique reward r sampled from $P(r|s, a)$ and $P(s, a)$ the unique next state s' sampled from $P(s'|s, a)$. We start by giving some background on the interplay of data structures and optimistic planning algorithms.

3.1. Data structures

In this work, we compare two data structures for planning in an MDP: tree and (directed) graph, represented in Figure 1. In order to distinguish them, we referring to trees with Roman symbols, e.g. T, U, L, B; and to graphs with calligraphic symbols, e.g. $\mathcal{G}, \mathcal{U}, \mathcal{L}, \mathcal{B}$. In both structures, we say that a node is *internal* if it has outgoing edges, and *external* else.

In a tree, a node of depth h represents a sequence of actions $a \in A^h$. The *root* of the tree corresponds to the empty action sequence, and hence to the initial state $s_0 \in S$. At iteration n, we denote the current tree as T_n. Borrowing notations from topology, we denote its set of internal nodes as $\mathring{T}_n$ and its set of external nodes (the leaves) as ∂T_n. Note that since the MDP is deterministic, a sequence of action a is associated with its final state denoted $s(a)$, but this association is not one-to-one: several sequences of action can lead to the same state, which will be represented several times in the tree.

In a graph, the nodes represent states $s \in S$, and the edges represent transitions between states. The *source* of the graph corresponds to the initial state s_0. At iteration n, we denote the current graph as $\mathcal{G}_n$, its set of internal nodes as $\mathring{\mathcal{G}}_n$ and its set of *sinks* as $\partial\mathcal{G}_n$.

Both structures are built iteratively from a single starting node, by selecting an external node (leaf or sink) to expand. The *expansion* of a node a or s refers to calling the generative model to sample the reward r and next state s' for each action $a \in A$, and adding child nodes to the data structure. In a tree, the expansion of a node $a \in A^h$ always lead to the

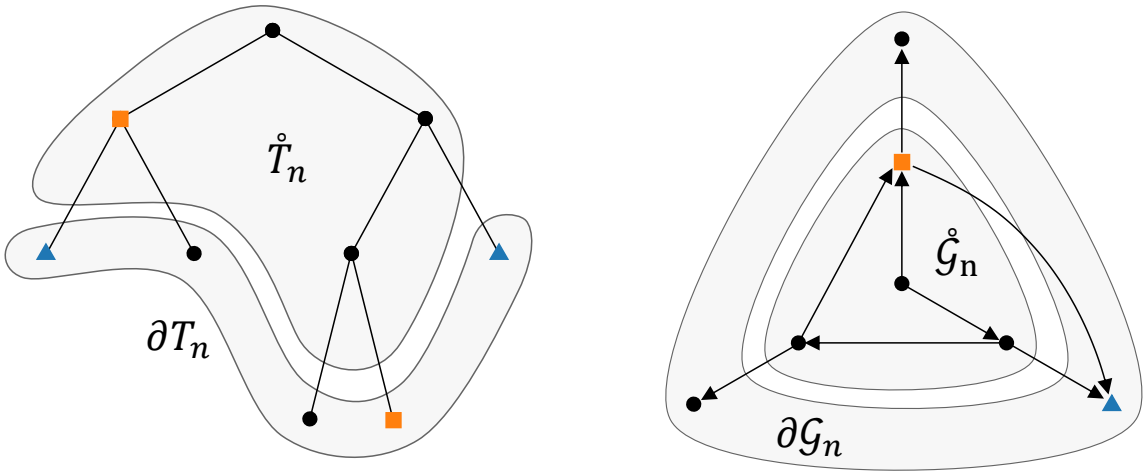


Figure 1: Illustration of the tree T_n (left) and the graph $\mathcal{G}_n$ (right) built from the same observed transitions. The root of the tree corresponds to the central graph node. In the tree, two nodes with the same colour and shape (not black round) lead to the same state.

creation of new leaves that represent the suffix sequence of action $ab \in A^{h+1}$, $b \in A$. The maximum depth of an expanded node in T_n is denoted d_n. In contrast, in a graph the next state s' reached from s, a might already be present in $\mathcal{G}_n$, in which case we add the edge between s and s' without creating a new node. These data structures can be used to store information about the MDP, such as the transitions and rewards $r(s, a)$, or other informations useful for planning.

3.2. Optimistic planning

A planning algorithm is typically composed of two main rules: (i) A *sampling rule*, that selects promising transitions to simulate at each iteration n; (ii) A *recommendation rule*, that recommends a good first action a_n to take (in s_0). These rules can be chosen with the goal of minimising the simple regret r_n. A popular approach is to follow the principle of *Optimism in the Face of Uncertainty* (OFU) (see Munos, 2014), which consists in exploring the option that maximises an upper-bound of the true objective. In the context of planning, it has been applied by forming bounds on the value function V.

Definition 1 (Value bounds) ***On trees.*** *We denote by $L : T_n \to \mathbb{R}$ and $U : T_n \to \mathbb{R}$ a lower-bound and upper-bound for the state value V defined on the tree T_n, such that*

$$\forall a \in T_n, \qquad L(a) \leq V(s(a)) \leq U(a).$$

In the sequel, we abuse notations and denote this inequality as $L \leq V \leq U$.

On graphs. *Likewise, we denote by $\mathcal{L} : \mathcal{G}_n \to \mathbb{R}$ and $\mathcal{U} : \mathcal{G}_n \to \mathbb{R}$ a lower-bound and upper-bound for the state value V defined on the graph $\mathcal{G}_n$, such that*

$$\forall s \in \mathcal{G}_n, \qquad \mathcal{L}(s) \leq V(s) \leq \mathcal{U}(s).$$

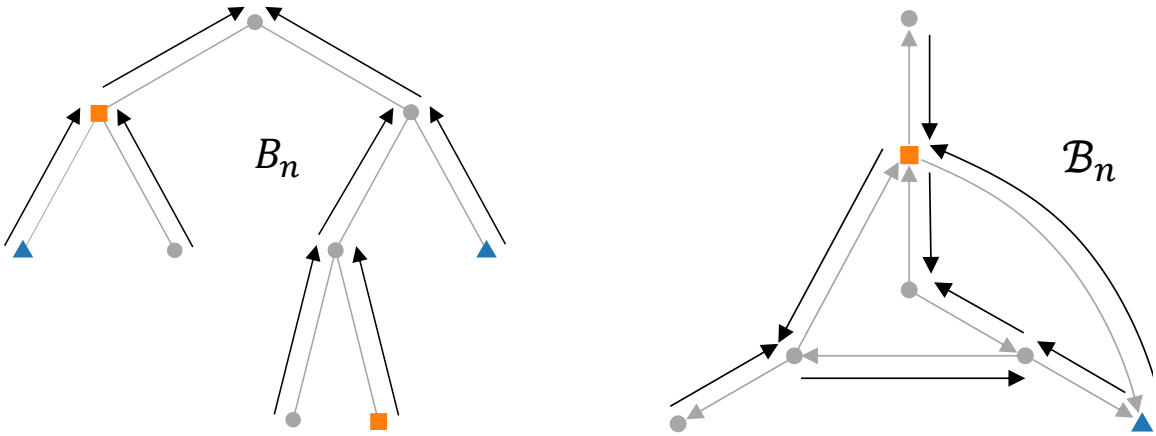


Figure 2: Black arrows depict how the Bellman backup operators B_n (left) and $\mathcal{B}_n$ (right) propagate value estimates from successor nodes to their parents. Information travels freely in a graph, but only upwards in a tree.

Following the OFU principle, at iteration n we must leverage available information to design an upper-bound U_n (or $\mathcal{U}_n$) on V as tight as possible. Then, in order to select a promising external node to expand, the sampling rule starts from the root (or source) and follows the optimistic strategy of always selecting the action which maximises U_n (or $\mathcal{U}_n$), until reaching an optimistic leaf (or sink) to expand. This strategy was used with great success in (e.g. Kocsis and Szepesvári, 2006; Hren and Munos, 2008; Bubeck and Munos, 2010; Busoniu and Munos, 2012).

For instance, since we assume that the rewards are bounded in $[0, 1]$, trivial bounds on $V(s)$ are $0 \leq V(s) \leq V_{\max} \overset{def}{=} \sum_t \gamma^t 1 = 1/(1-\gamma)$. But these trivial bounds are the same for every node, which makes them non-informative, and do not make use of the observed information. However, they can be used as a valid starting point. Each observed transition can then be used to tightened these bounds, by resorting to the Bellman optimality operator.

Definition 2 (Bellman optimality operator) ***On trees.*** *We define the Bellman optimality operator B_n on the tree T_n as:*

$$B_n(f)(a) \overset{def}{=} \begin{cases} \max_{b \in A} r(s(a), b) + \gamma f(ab) & \text{if } a \in \mathring{T}_n; \\ f(a) & \text{if } a \in \partial T_n; \end{cases} \tag{2}$$

where $f : T_n \to \mathbb{R}$ is a real-valued function of tree nodes, such as the value bounds L, U.

On graphs. *Likewise, we define the Bellman optimality operator $\mathcal{B}_n$ on the graph $\mathcal{G}_n$:*

$$\mathcal{B}_n(f)(s) \overset{def}{=} \begin{cases} \max_{b \in A} r(s, b) + \gamma f(P(s, b)) & \text{if } s \in \mathring{\mathcal{G}}_n; \\ f(s) & \text{if } s \in \partial \mathcal{G}_n. \end{cases} \tag{3}$$

The updates with both Bellman operators are depicted in Figure 2.

Hren and Munos (2008) used this Bellman operator B_n in their `OPD` algorithm to define a pair of bounds (L_n, U_n) at each iteration n. They use trivial bounds at the leaves, and

backup these estimates up to the root by iteratively applying B_n. We can show that, under a *monotonicity* condition (satisfied by the trivial bounds 0 and V_{max}), applying B_n can only tighten a bound and converges in a finite time.

Definition 3 (Monotonicity) *A pair of bounds (L, U) or $(\mathcal{L},\mathcal{U})$ is* monotonic *if they are respectively non-decreasing and non-increasing along transitions:*

$$\begin{aligned} \forall a \in T_n, \quad & L(a) \leq B_n(L)(a), && U(a) \geq B_n(U)(a) \\ \forall (s) \in \mathring{\mathcal{G}}_n, \quad & \mathcal{L}(s) \leq \mathcal{B}_n(\mathcal{L})(s), && \mathcal{U}(s) \geq \mathcal{B}_n(\mathcal{U})(s) \end{aligned}$$

Lemma 4 (Properties of B_n) *(i) B_n preserves monotonicity and tightens monotonic bounds:*

$$\text{if } L \leq V \leq U, \text{ then } L \leq B_n(L) \leq V \leq B_n(U) \leq U;$$

(ii) The sequence $B_n^k = \underbrace{B_n \circ \cdots \circ B_n}_{k \text{ times}}$ converges in a finite time $k = d_n$, where d_n is the depth of T_n.

This enables Hren and Munos (2008) to define[1] non-trivial valid bounds on V:

$$L_n \overset{def}{=} B_n^{d_n}(0), \qquad U_n \overset{def}{=} B_n^{d_n}(V_{\max}), \tag{4}$$

where 0 is the null function. The corresponding `OPD` algorithm is described in Algorithm 1.

Algorithm 1: The *Optimistic Planning of Deterministic Systems* (`OPD`) algorithm from (Hren and Munos, 2008).

for *each iteration n* **do**
  Compute the bounds $L_n = B_n^{d_n}(0)$ and $U_n = B_n^{d_n}(V_{\max})$.
  $b_n \leftarrow \emptyset$
  while *the node $b_n \in \mathring{T}_n$ is internal* **do**
    $b_n \leftarrow \arg\max_{a' \in b_n A} r(a') + \gamma U_n(a')$ ▷ `Optimistic sampling rule`
  for *action $a \in A$* **do** ▷ `Node expansion`
    Simulate $r \leftarrow r(s(b_n), a)$ and $s' \leftarrow P(s(b_n), a)$.
    Add a new leaf $b_n a$ to T_{n+1}, with associated reward r.
return $\arg\max_{a \in A} r(s,a) + \gamma L_n(a)$. ▷ `Conservative recommendation rule`

Likewise, we show that the graph version $\mathcal{B}_n$ verifies similar properties.

Lemma 5 (Properties of $\mathcal{B}_n$) *(i) $\mathcal{B}_n$ preserves monotonicity and tightens monotonic bounds:*

$$\text{if } \mathcal{L} \leq V \leq \mathcal{U}, \text{ then } \mathcal{L} \leq \mathcal{B}_n(\mathcal{L}) \leq V \leq \mathcal{B}_n(\mathcal{U}) \leq \mathcal{U};$$

(ii) $\mathcal{B}$ is a γ-contraction, and we denote $\mathcal{B}_n^\infty \overset{def}{=} \lim_{k\to\infty} \mathcal{B}_n^k$.

This motivates us to propose Algorithm 2, following the approach of Algorithm 1 adapted to a graph structure.

1. We use an iteration of operators while a recursive definition was used originally.

Algorithm 2: Our proposed *Graph-Based Optimistic Planning for Deterministic systems* (GBOP-D) algorithm.

for *each iteration* n **do**

1 Compute the bounds $\mathcal{L}_n = \mathcal{B}_n^\infty(0)$ and $\mathcal{U}_n = \mathcal{B}_n^\infty(V_{\max})$.

$s_n \leftarrow s_0$

while *the node* $s_n \in \mathring{\mathcal{G}}_n$ *is internal* **do**

2 $s_n \leftarrow \arg\max_{s'} r(s_n, a) + \gamma \mathcal{U}_n(s')$ ▷ `Optimistic sampling rule`

for *action* $a \in A$ **do** ▷ `Node expansion`

Simulate $r \leftarrow r(s_n, a)$ and $s' \leftarrow P(s_n, a)$.

Get or create the node s' in $\mathcal{G}_{n+1}$, and add the transition $(s_n, a) \to s', r$.

return $\arg\max_{a \in A} r(s, a) + \gamma \mathcal{L}_n(s(a))$. ▷ `Conservative recommendation rule`

Remark 6 (Termination and complexity) *One key difficulty both in the design and analysis of the algorithm is the correct handling of loops in the graph. Indeed, there are two procedures in GBOP-D that may not terminate in finite time whenever $\mathcal{G}_n$ contains a loop: the computation of $\mathcal{B}_n^\infty$ (line 1) and the sampling rule loop (line 2). We handle these steps carefully in the Supplementary Material, where we discuss an approximate implementation in which these two procedures are stopped whenever they reach a desired accuracy ε, along with an analysis of the corresponding time complexity and impact on the performance.*

Though both algorithms share a similar design, we claim that using graphs provides substantial theoretical and practical performance improvements, and back up this statement in Sections 4 and 6.

4. Analysis

Comparing OPD and GBOP-D directly is difficult since they do not involve the same structure, which implies implicit differences in their behaviours. Studying them under a common framework makes these differences explicit. To leverage the analysis of OPD by Hren and Munos (2008), we will frame GBOP-D as a tree-based planning algorithm: the graph operator $\mathcal{B}$ will be represented as tree backup B applied on an *unrolled* tree $T(\mathcal{G}_n)$, defined below.

4.1. Background on the sample complexity of OPD

First, we recall the analysis of Hren and Munos (2008) and introduce some notations.

Definition 7 (Sequence values) *The value of a finite* ***sequence*** *of actions $a \in A^h$ is:*

$$V(a) = R(s_0, a) + \gamma^h V(s(a)),$$

where $R(s, a) = \sum_{t=0}^{h-1} \gamma^t r_t$ is the return obtained by executing the sequence of actions a starting from the state s. We also denote $V^\star = \max_{a \in A} V(a)$ the value of the best action.

This enables to define a measure of the difficulty of a planning problem.

Definition 8 (Difficulty measure) *We define the near-optimal branching factor* $\boldsymbol{\kappa}$ *of an MDP as*

$$\boldsymbol{\kappa} = \limsup_{h\to\infty} |\boldsymbol{T}_h^\infty|^{1/h} \in [1, K] \tag{5}$$

where $T_h^\infty = \left\{ a \in A^h : V^\star - V(a) \leq \frac{\gamma^h}{1-\gamma} \right\}$ *is the set of near-optimal nodes at depth* h.

This problem-dependent measure $\boldsymbol{\kappa}$ is the branching factor of the subtree $T^\infty = \bigcup_h T_h^\infty$ of near-optimal nodes that can be sampled by OPD, and acts as an effective branching factor as opposed to the true branching factor K. When $\boldsymbol{\kappa}$ is small, fewer nodes must be explored at a given depth allowing the algorithm to plan deeper for a given budget n. Thus, it directly impacts the simple regret that can be achieved by OPD when run on a given MDP.

Theorem 9 (Regret bound of Hren and Munos 2008) *The Algorithm 1 enjoys the following regret bound:*

$$r_n = \widetilde{\mathcal{O}}\left(n^{-\log\frac{1}{\gamma}/\log\boldsymbol{\kappa}}\right),$$

where $f_n = \widetilde{\mathcal{O}}(n^{-\alpha})$ *means that for any* $\alpha' < \alpha$, $f_n = \mathcal{O}(n^{-\alpha'})$, *for all* $\alpha \in \mathbb{R}_+ \cup \{+\infty\}$.

The near-optimal branching factor $\boldsymbol{\kappa}$ is related (Bubeck and Munos, 2010) to the near-optimality dimension studied in the online optimisation literature (see e.g. Bubeck et al., 2009; Munos, 2011). It is typically small in problems where there is one single optimal trajectory, of which any deviation can be quickly dismissed as suboptimal. Conversely, κ is large when many sub-optimal trajectories cannot be distinguished easily based on their values, which requires the exploration of a large part of the tree T of branching factor K.

4.2. Motivation for an improved regret bound

We start by reformulating the sampling rule used for the OPD algorithm. To that end, notice that when some bounds (L, U) on the state values $V(s(a))$ are available, they also induce bounds $(\overline{L}, \overline{U})$ on values $V(a)$ of sequences of actions a of length h defined as:

$$\underbrace{R(s_0,a) + \gamma^h L(a)}_{\overline{L}(a)} \leq V(a) \leq \underbrace{R(s_0,a) + \gamma^h U(a)}_{\overline{U}(a)}.$$

One can easily see that, since the (L_n, U_n) used in the optimistic sampling rule described in Algorithm 1 are invariant by B_n by definition, this rule can be equivalently expressed as:

$$b_n \in \arg\max_{a\in\partial T_n} \overline{U}_n(a). \tag{6}$$

Likewise, the conservative recommendation rule returns the first action of:

$$a_n \in \arg\max_{a\in\partial T_n} \overline{L}_n(a) \tag{7}$$

As shown in Figure 2, in a tree the Bellman operator B_n only propagates the information upward, and the leaves cannot be updated. Thus, $U_n = B_n^{d_n}(V_{\max})$ and $V_{\max}$ coincide on

∂T_n which means that the sampling rule of OPD can be summarized as using (6) with the trivial upper-bound $U_n = V_{\max}$. Likewise, the recommendation rule simply uses (7) with the trivial lower-bound $L_n = 0$. Thus, OPD amounts to simply using the trivial bound $(0, V_{\max})$ on leaf nodes, and does not make use of all the available information in T_n to improve these bounds.

Let us now assume for the moment that we have access to tighter bounds (L, U) provided by an oracle:

$$0 \leq L \leq V \leq U \leq V_{\max}.$$

Definition 10 (A finer difficulty measure) *We define the near-optimal branching factor* according to the bounds (L, U) *as*

$$\boldsymbol{\kappa(L,U) \overset{def}{=} \limsup_{h\to\infty} \left|T_h^\infty(L,U)\right|^{1/h} \in [1,K],} \tag{8}$$

where $T_h^\infty(L,U) = \left\{a \in A^h : V^\star - V(a) \leq \gamma^h (U(a) - L(a))\right\}$.

Lemma 11 *This branching factor shrinks as the bounds (L, U) get tighter:*

$$L_2 \leq L_1 \leq V \leq U_1 \leq U_2 \implies \kappa(L_1,U_1) \leq \kappa(L_2,U_2).$$

In particular, $\boldsymbol{\kappa(L,U) \leq \kappa}$.

Theorem 12 *Let $L \leq V \leq U$ monotonic bounds, then planning with L and U in (6) and (7) yields the following simple regret bound:*

$$r_n = \widetilde{\mathcal{O}}\left(n^{-\log\frac{1}{\gamma}/\log \boldsymbol{\kappa(L,U)}}\right).$$

This theorem states that we can potentially improve the performance of the planning algorithm if we manage to find bounds (L, U) that are tighter than the trivial ones at the leaves ∂T_n, which may be possible if we have already seen the states corresponding to this leaves, but it does not explain how to obtain such bounds. In the next subsection, we describe a method to build a sequence of increasingly tight bounds (L_n, U_n), at each planning iteration n. The corresponding regret bound, our main result, is stated in Theorem 16.

4.3. Unrolling the tree to tighten the bounds

In order to reproduce the behaviour of Algorithm 2 on a tree structure, we rely on the following observation: expanding a node s in $\mathcal{G}_n$ simultaneously expands all the paths leading to this node. To account for this observation in the analysis, we will consider an *unrolling* operator T, illustrated in Figure 3, that builds a potentially infinite tree $T(\mathcal{G}_n)$ containing every sequence of action that can be traversed in a graph $\mathcal{G}_n$.

$$T(\mathcal{G}_n) = \{a \in A^h : s_{t+1} \in \mathcal{G}_n \text{ with } s_{t+1} = P(s_t, a_t) \text{ for } 0 \leq t < h\} \tag{9}$$

We analyse GBOP-D though the prism of $T(\mathcal{G}_n)$, which is only used as a theoretical tool.

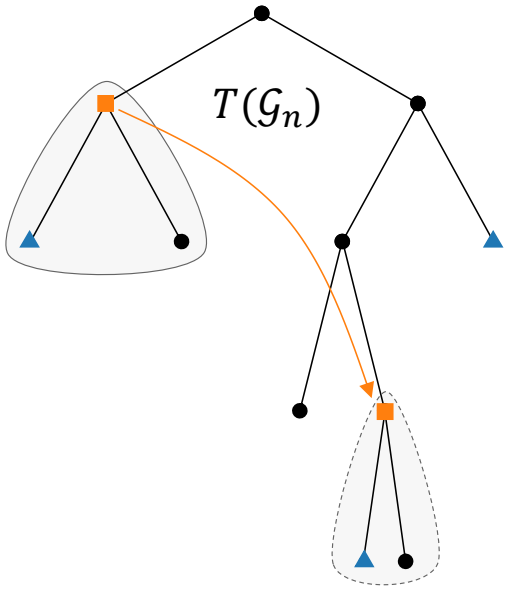


Figure 3: The tree $T(\mathcal{G}_n)$ obtained by unrolling $\mathcal{G}_n$. Contrary to T_n shown in Figure 1, the orange leaf a is expanded at the same time as the internal orange node, which enables to tighten its value bounds $(L_n(a), U_n(a))$ by applying B_n.

We can define the counterpart of the bounds $(\mathcal{L}_n, \mathcal{U}_n)$ in the same way as (4) applied to $T(\mathcal{G}_n)$ rather than T_n, except that the depth d_n of $T(\mathcal{G}_n)$ can now be infinite:

$$L_n = B_n^\infty(0), \qquad U_n = B_n^\infty(V_{\max}). \tag{10}$$

This definition is equivalent to that of GBOP-D in the sense that:

Lemma 13 (Bound equivalence) *For any sequence of action $a \in T(\mathcal{G}_n)$, we have $L_n(a) = \mathcal{L}_n(s(a))$ and $U_n(a) = \mathcal{U}_n(s(a))$.*

In $T(\mathcal{G}_n)$, the unrolling mechanics behave as if any leaf a sharing the same state $s(a)$ as an internal node a' was automatically expanded, and thus had its bound $L_n(a), U_n(a)$ tightened by the Bellman backup B_n to a sub-interval of the trivial bounds $(0, V_{\max})$ that are used in OPD.

The sampling and recommendation rules of GBOP-D amount to running those of OPD on the tree $T(\mathcal{G}_n)$, except that the sampled sequence b_n and recommended sequence a_n can now have infinite depth since $T(\mathcal{G}_n)$ itself can be infinite (we say that a_n and b_n are represented by nodes of infinite depth). In the sequel, we analyse how these rules behave on $T(\mathcal{G}_n)$.

Lemma 14 (Expansion) *Any node a of depth h traversed by the optimistic sampling rule of GBOP-D at iteration n belongs to $T_h^\infty(L_n, U_n)$:*

$$V^\star - V(a) \leq \gamma^h (U_n(a) - L_n(a)). \tag{11}$$

In particular, if the sampling rule samples an infinite sequence $a \in A^\infty$, it is an optimal sequence, and we write that (2) also holds for a with $h = \infty$.

Lemma 15 (Recommendation) *The action a_n recommended by GBOP-D has a simple regret $r_n \leq \frac{\gamma^{d_n}}{1-\gamma}$, where $d_n \in \mathbb{R} \cup \{\infty\}$ is the maximal depth of expanded nodes in $T(\mathcal{G}_n)$.*

Note that even though $T(\mathcal{G}_n)$ can be infinite, there is only one node b_t that is selected for expansion at each iteration $t \leq n$.

4.4. Regret guarantee

In Theorem 12, we assumed that some bounds (L, U) were revealed by an oracle and available from the onset for planning. In (10), we instead built a *sequence* of bounds $(L_n, U_n)_{n\geq 0}$ (10) that is non-increasing in the sense of inclusion, i.e. $0 \leq \cdots \leq L_{n-1} \leq L_n \leq V \leq U_n \leq U_{n-1} \leq \cdots \leq V_{\max}$.

We can consider the sequence $\kappa_n = \kappa(L_n, U_n)$. By Lemma 11, it is non-increasing and lower-bounded by 1, thus converges. Let $\boldsymbol{\kappa_\infty = \lim_{n\to\infty} \kappa(L_n, U_n)} \in [1, K]$.

Theorem 16 *GBOP-D enjoys the following regret bound, with $\kappa_\infty \leq \kappa$:*

$$r_n = \widetilde{\mathcal{O}}\left(n^{-\log \frac{1}{\gamma}/\log \kappa_\infty}\right).$$

Intuitively, κ_∞ should be much lower than κ in problems where trajectories overlap a lot. For instance, it will be the case when two actions cancel each-other out (e.g. moving left or right), or are commutative (e.g. placing pawns on a board game). However, this is merely an intuition. We now show that there exist problem instances in which $\kappa_\infty < \kappa$, which is a legitimate concern since their non-existence would make Theorem 16 trivial.

4.5. Illustrative example

In Proposition 17, we consider a toy MDP $\mathcal{M}$ shown in Figure 4. The transitions are described visually while the rewards are defined as follows: let $0 \leq r^\star \leq \gamma$, and $r^- = r^\star - \frac{\gamma}{1-\gamma} S$, $r^+ = r^\star + S$ with $S = r^\star \left(\frac{1}{\gamma} - 1\right)$. Note that this choice ensures that $r^\star, r^-, r^+$ and S are all in $[0, 1]$.

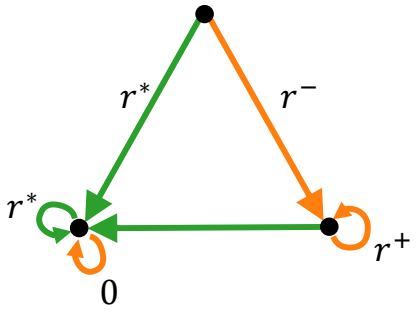


Figure 4: A toy MDP with three states and $K \geq 2$ actions. We start in the top state. The first action a_1 is represented by green arrows, and all other actions $a_2, \ldots, a_K$ are represented by orange arrows. The rewards are shown next to the transitions.

Proposition 17 (Branching factors) *The MDP $\mathcal{M}$ verifies $\boldsymbol{\kappa = K - 1}$ and $\boldsymbol{\kappa_\infty = 1}$.*

This result confirms that Theorem 16 is non-trivial since we exhibit a problem for which $\kappa_\infty < \kappa$ (when $K \geq 3$), and legitimates our attempt to improve planning performances by merging the tree into a graph.

5. Extension to Stochastic Systems

The approach developed in Sections 3 and 4 consists in using state similarity to tighten a pair of lower and upper bounds (L, U) for the value function V. Thus, any planning algorithm that is based on such bounds can benefit from this insight, and any theoretical result based on the validity and rate of convergence of these bounds will be preserved.

Confidence intervals for rewards. When the reward distribution $P(r \mid s, a)$ is stochastic, deviation inequalities can be used to design a confidence interval $[\ell_t(s,a), u_t(s,a)]$ over its expected value $\mathbb{E}[r|s,a]$. For instance, the Chernoff-Hoeffding deviation inequality was used to design confidence intervals in (Kocsis and Szepesvári, 2006; Bubeck and Munos, 2010; Kaufmann and Koolen, 2017). In recent works (Leurent and Maillard, 2019; Jonsson et al., 2020), the tighter Kullback-Leibler confidence interval is preferred:

$$u_t(s,a) \overset{def}{=} \max\left\{v : \mathrm{kl}\big(\hat{r}_t(s,a), v\big) \leq \frac{\beta^r(n_t(s,a), n)}{n_t(s,a)}\right\},$$
$$\ell_t(s,a) \overset{def}{=} \min\left\{v : \mathrm{kl}\big(\hat{r}_t(s,a), v\big) \leq \frac{\beta^r(n_t(s,a), n)}{n_t(s,a)}\right\},$$

where $n_t(s,a)$ is the number of times the transition (s,a) was visited, $\hat{r}_t(s,a)$ is the empirical mean reward, β^r is an exploration function and $\mathrm{kl}(u,v)$ is the binary Kullback-Leibler divergence between Bernoulli distributions: $\mathrm{kl}(u,v) = u \log \frac{u}{v} + (1-u) \log \frac{1-u}{1-v}$.

Confidence region for transitions. Likewise, when the transition distribution $P(s'|s,a)$ is stochastic, a confidence set on the probability vector $p(\cdot|s,a)$ can be defined as $\mathcal{C}_t(s,a) \overset{def}{=} \left\{p \in \Sigma_S : \mathrm{KL}\big(\widehat{p}_t(\cdot|s,a), p\big) \leq \frac{\beta^p(n_t(s,a),n)}{n_t(s,a)}\right\}$, where $\widehat{p}_t(\cdot|s,a) \overset{def}{=} n_t(s,a,\cdot)/n_t(s,a)$ is the empirical distribution, Σ_S is the probability simplex over S, β^p is an exploration function and $\mathrm{KL}(p,q) = \sum_{s \in S} p(s) \log \frac{p(s)}{q(s)}$ is the Kullback-Leibler divergence between categorical distributions.

Bellman operator with stochasticity. In this work, we do not discuss the tuning of β^r, β^p, but simply assume that they are chosen such that the rewards and transitions belong to their confidence regions with sufficiently high probability to obtain performance guarantees for the planning algorithm. For more details on such a choice, refer to (e.g. Leurent and Maillard, 2019; Jonsson et al., 2020). We modify the Definition 2 of $\mathcal{B}_t$ as:

$$\mathcal{B}_t^+(\mathcal{U})(s) = \max_{a \in A}\left[u_t(s,a) + \gamma \max_{q \in \mathcal{C}_t(s,a)} \sum_{s'} q(s'|s,a)\mathcal{U}(s')\right],$$
$$\mathcal{B}_t^-(\mathcal{L})(s) = \max_{a \in A}\left[\ell_t(s,a) + \gamma \min_{q \in \mathcal{C}_t(s,a)} \sum_{s'} q(s'|s,a)\mathcal{L}(s')\right],$$

for all $s \in \mathring{\mathcal{G}}_n$, where the extremums over these KL confidence regions $\mathcal{C}_t(s,a)$ can be computed as explained in (Filippi et al., 2010, Appendix A). Under the event that all confidence regions $[\ell_t(s,a), u_t(s,a)]$ and $\mathcal{C}_t(s,a)$ are valid, the Lemma 5 still holds for $\mathcal{B}_t^-, \mathcal{B}_t^+$.

Structure of the planning algorithm In the deterministic setting, once a transition has been observed, it is known with certainty and doesn't need to be sampled ever again, which is why only external nodes $\partial\mathcal{G}_n$ are sampled in `GBOP-D`. Conversely, in the stochastic setting the expected reward and transition probabilities must be estimated from samples, which implies that internal nodes $\mathring{\mathcal{G}}_n$ must be sampled as well. Then, it is common to adopt and episodic setting where we sample trajectories of a fixed horizon H, tuned depending on the budget n. This is the case in (e.g. Kearns et al., 2002; Kocsis and Szepesvári, 2006; Bubeck and Munos, 2010; Feldman and Domshlak, 2014; Leurent and Maillard, 2019; Jonsson et al., 2020). We also follow this scheme in our proposed `GBOP` algorithm.

Algorithm 3: *Graph-Based Optimistic Planning* (`GBOP`) algorithm.

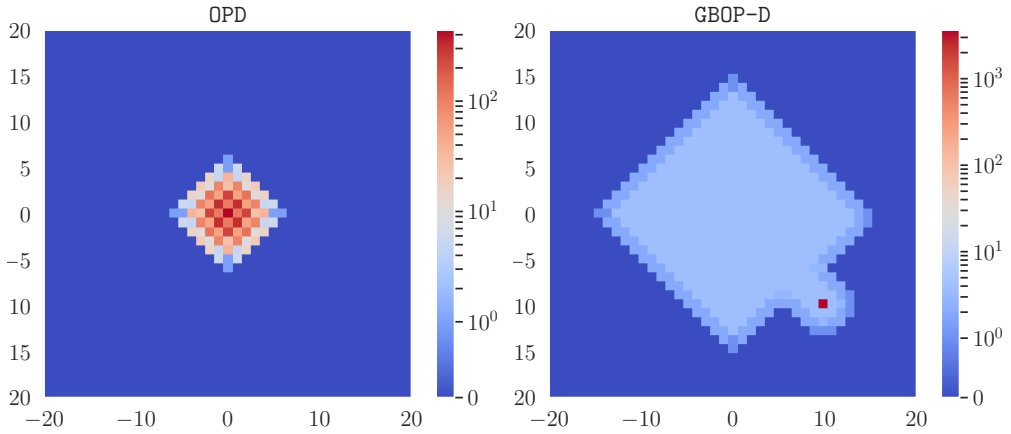
for *trajectory* m *in* $[1, M]$ **do**
  for *time* t *in* $[1, H]$ **do**
    $n \leftarrow (m-1)H + t$.
    Compute the bounds $\mathcal{L}_n = (\mathcal{B}_n^-)^\infty(0)$ and $\mathcal{U}_n = (\mathcal{B}_n^+)^\infty(V_{\max})$.
    $b_t \leftarrow \arg\max_{a \in A} r(s_t, a) + \gamma \mathcal{U}_n(s')$ ▷ `Optimistic sampling rule`
    Simulate $r_t, s_{t+1} \sim P(r, s_{t+1} \mid s_t, b_t)$. Get or create the node s_{t+1} in $\mathcal{G}_{n+1}$, and add an occurrence of the transition (s_t, b_t, r_t, s_{t+1}).
return $\arg\max_{a \in A} r(s,a) + \gamma \mathcal{L}_n(s(a))$. ▷ `Conservative recommendation rule`

6. Numerical Illustration

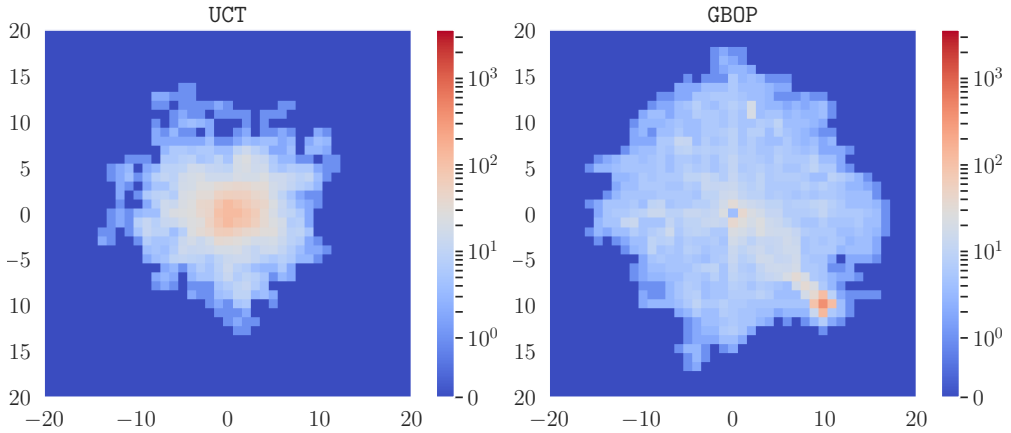
To evaluate the practical benefits of our approach, we compare graph-based and tree-based planning algorithms on two domains.

Gridworld domain. We consider a grid in which the agent starts at $(0,0)$ and can move in $K = 4$ directions. The reward function is 0 everywhere, except in the vicinity of a goal located at $(10,10)$, around which the reward decreases quadratically from 1 to 0 in a ball of radius 5. The Figure 5(a) shows number of times a state is sampled by `OPD` and `GBOP-D`, both run with a budget $n = 5460$ and discount $\gamma = 0.95$. In the absence of rewards, `OPD` samples sequences of actions uniformly (in a breadth-first search manner), which –because of the dynamics structure– results in a non-uniform occupancy of the state space S, where the trajectories concentrate near the starting state. In contrast, `GBOP-D` explores uniformly in S, sampling each state up to four times (from its four neighbours), until it finds the goal vicinity and finally samples the goal location indefinitely. We reproduce the experiment in the stochastic setting by adding noise on the transitions with probability $p = 10\%$, and comparing `GBOP` to `UCT` as we show in Figure 5(b). To quantify these qualitative differences, we define in Figure 5(c) an exploration score: the mean distance $d(s_t, s_0)$ of sampled states to the initial state (exploration) minus the distance $d(s_t, s_g)$ to the goal state (exploitation).

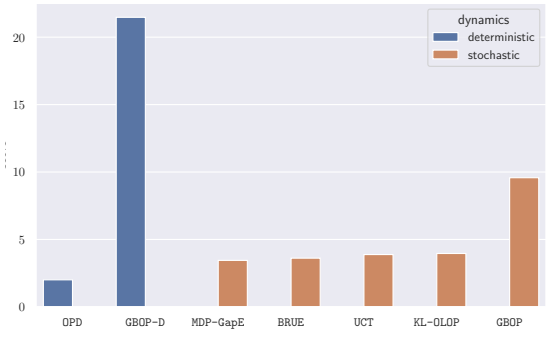
Sailing domain (Vanderbei, 1996). In a second experiment, a boat is sailing in $K = 8$ directions to reach a goal, and suffers a cost (move duration) that depends on the direction



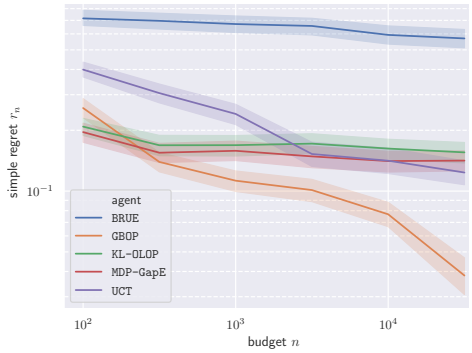
(*a*) State occupancies in a deterministic gridworld.



(*b*) State occupancies in a stochastic gridworld.



(*c*) Exploration scores in the gridworld domains.



(*d*) Simple regret r_n in the sailing domain.

Figure 5: Benchmark of planning performances.

of the wind which follows stochastic dynamics. Figure 5(*d*) shows the evolution of the simple regret r_n of stochastic planning algorithms with respect to the number n of oracle

calls. We compute the mean regret and its 95% confidence interval over 500 simulations. The asymptotic log-log slope σ provides an empirical measurement of the effective branching factor $\kappa_e = \exp(-\log(1/\gamma)/\sigma)$ for each algorithm. We measure that $\sigma \approx -0.04$ and $\kappa_e \approx 3.6$ for `BRUE`, `KL-OLOP`, `MDP-GapE`, `UCT`. In contrast, we measure $\sigma \approx -0.3$ and $\kappa_e \approx 1.2$ for `GBOP`, which suggests that our result of Theorem 16 might generalize to the stochastic setting. Additional experimental results and details are provided in the Supplementary Material.

Acknowledgement

This work was supported by the French Ministry of Higher Education and Research, and CPER Nord-Pas de Calais/FEDER DATA Advanced data science and technologies 2015-2020. We thank Mathieu Seurin and Émilie Kaufmann for their valuable comments.

Conclusion

In this paper, we study a simple yet highly effective variant of the tree-based planning strategies. We prove that leveraging the graph structure induced by states provides a benefit over tree-based algorithms, in the form of an improved regret bound in the deterministic setting, that depends on a smaller difficulty measure. This translates into an enhanced performance in practice, and can be adapted to stochastic planning problems as we show empirically. We believe that revisiting the heart of the MCTS strategy to take into account a graph structure opens exciting novel research directions.

References

Joaquin Ballesteros, Luis Merino, Miguel Angel Trujillo, Antidio Viguria, and Anibal Ollero. Improving the efficiency of online POMDPs by using belief similarity measures. In *Proc. of ICRA*, 2013.

S. Bubeck and R. Munos. Open loop optimistic planning. In *Proc. of COLT*, 2010.

Sébastien Bubeck, Gilles Stoltz, Csaba Szepesvári, and Rémi Munos. Online optimization in x-armed bandits. In *Advances in NIPS*, 2009.

Lucian Busoniu and Rémi Munos. Optimistic planning for markov decision processes. In *Artificial Intelligence and Statistics*, pages 182–189, 2012.

P.-A. Coquelin and R. Munos. Bandit Algorithms for Tree Search. *Proc. of UAI*, 2007.

Rémi Coulom. Efficient Selectivity and Backup Operators in Monte-Carlo Tree Search. In *Proc. of ICCG*, 2006.

Zohar Feldman and Carmel Domshlak. Simple regret optimization in online planning for markov decision processes. *Journal of Artifial Intelligence Research*, 51:165–205, 2014.

S. Filippi, O. Cappé, and A. Garivier. Optimism in Reinforcement Learning and Kullback-Leibler Divergence. In *Allerton Conference on Communication, Control, and Computing*, 2010.

J.-B. Grill, M. Valko, and R. Munos. Blazing the trails before beating the path: Sample-efficient monte-carlo planning. In *Advances in NIPS*, 2016.

Jean-Bastien Grill, Omar Darwiche Domingues, Pierre Ménard, Rémi Munos, and Michal Valko. Planning in entropy-regularized markov decision processes and games. In *Advances in NeurIPS*, 2019.

Jesse Hostetler, Alan Fern, and Tom Dietterich. State aggregation in monte carlo tree search. In *Proc. of AAAI*, 2014.

Jean-Francois Hren and Rémi Munos. Optimistic planning of deterministic systems. In *Proc. of EWRL*, 2008.

Ruitong Huang, Mohammad M. Ajallooeian, Csaba Szepesvári, and Martin Müller. Structured best arm identification with fixed confidence. In *Proc. of ALT*, 2017.

Anders Jonsson, Emilie Kaufmann, Pierre Ménard, Omar Darwiche Domingues, Edouard Leurent, and Michal Valko. Planning in markov decision processes with gap-dependent sample complexity, 2020.

Emilie Kaufmann and Wouter M Koolen. Monte-carlo tree search by best arm identification. In *Advances in NIPS*, pages 4897–4906, 2017.

Michael J. Kearns, Yishay Mansour, and Andrew Y. Ng. A sparse sampling algorithm for near-optimal planning in large markov decision processes. *Machine Learning*, 49(2-3): 193–208, 2002.

Levente Kocsis and Csaba Szepesvári. Bandit based monte-carlo planning. In *Proc. of ECML*, 2006.

Edouard Leurent and Odalric-Ambrym Maillard. Practical open-loop optimistic planning. In *Proc. of ECML-PKDD*, 2019.

R. Munos. *From bandits to Monte-Carlo Tree Search: The optimistic principle applied to optimization and planning.* Foundations and Trends in Machine Learning, 2014.

Rémi Munos. Optimistic optimization of a deterministic function without the knowledge of its smoothness. In *Advances in NIPS*, 2011.

David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy P. Lillicrap, Karen Simonyan, and Demis Hassabis. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362, 2018.

B. Szorenyi, G. Kedenburg, and R. Munos. Optimistic planning in markov decision processes using a generative model. In *Advances in NIPS*, 2014.

Robert Vanderbei. Optimal sailing strategies, statistics and operations research program. https://vanderbei.princeton.edu/sail/sail.html, 1996.