Continuous-action planning for discounted infinite-horizon nonlinear optimal control with Lipschitz values [⋆]

Lucian Busoniu [a] Előd Páll [a] Rémi Munos [b]

[a] *Automation Department, Technical University of Cluj-Napoca, Romania*

[b] *Google DeepMind, London, UK*

Abstract

We consider discrete-time, infinite-horizon optimal control problems with discounted rewards. The value function must be Lipschitz continuous over action (input) sequences, the actions are in a scalar interval, while the dynamics and rewards can be nonlinear/nonquadratic. Exploiting ideas from artificial intelligence, we propose two optimistic planning methods that perform an adaptive-horizon search over the infinite-dimensional space of action sequences. The first method optimistically refines regions with the largest upper bound on the optimal value, using the Lipschitz constant to find the bounds. The second method simultaneously refines all potentially optimistic regions, without explicitly using the bounds. Our analysis proves convergence rates to the global infinite-horizon optimum for both algorithms, as a function of computation invested and of a measure of problem complexity. It turns out that the second, simultaneous algorithm works nearly as well as the first, despite not needing to know the (usually difficult to find) Lipschitz constant. We provide simulations showing the algorithms are useful in practice, compare them with value iteration and model predictive control, and give a real-time example.

Key words: Optimal control; planning; nonlinear systems; near-optimality analysis.

1 Introduction

We consider optimal control problems that require maximizing a discounted sum of rewards (the value), along an infinitely long discrete-time trajectory of the system. Such problems are encountered in automatic control [12] as well as in many other fields, including artificial intelligence (AI) [20], operations research, medicine, economics, etc. When the system and reward function have a general form, the problem must be solved approximately with numerical algorithms. A popular class of techniques is approximate dynamic programming [1], which computes offline a near-optimal value function and a state feedback control. Because it searches for a global solution, the complexity of dynamic programming usually grows fast with the state dimensionality [1].

We focus instead on receding-horizon algorithms that remove the direct dependence on the state space size, at the cost of solving a new problem at each step, locally for the current state of the system. A sequence of actions (inputs) is obtained, the initial action of this sequence is applied, and the procedure is repeated online for subsequent states. In automatic control, this is called receding-horizon model predictive control (MPC) [7], while in AI it is called online planning [14]. Note that computation still grows with the action space size and with the search horizon.

We search over the space of infinitely long sequences, using the optimistic planning (OP) class of algorithms [18]. OP methods originate in AI and perform a branch-and-bound search over the sequences, refining the region with the best upper bound on the value – hence the "optimistic" label. The main strengths of OP are the generality of the dynamics and rewards addressed, and a tight relation between computation and near-optimality, obtained using ideas from bandit theory and reinforcement learning. Many OP variants have been proposed for discrete actions, e.g. [13,10,16]. Our major aim in this paper is to address instead continuous actions, since they are essential in control.

Specifically, we propose two *optimistic planning algorithms with continuous actions* (OPC) that work in systems with general nonlinear dynamics and scalar, compact actions. The methods iteratively split the infinite-dimensional hyperrectangle of continuous-action sequences into smaller hyperrectangles (boxes), leading to an adaptive search horizon. They rely on a central Lipschitz property of the value function over action sequences, which is satisfied e.g. when the dynamics and rewards are Lipschitz, with a small enough constant for the dynamics. Using this property, an upper bound on the optimal value is derived using the rewards and size of the box. This leads to the first algorithm, which optimistically selects for splitting the box with the largest upper bound, and so it is called simply *OPC*. An essen-

[⋆] Corresponding author L. Busoniu.

Email addresses: `lucian@busoniu.net` (Lucian Busoniu), `pall.elod@gmail.com` (Előd Páll), `munos@google.com` (Rémi Munos).

tial insight is that each dimension k contributes to the bound with weight γ^k, where γ is the discount factor, and this is used to select which box dimension to split. We characterize the rate at which the box size shrinks with the number of splits, and define a measure of problem complexity, in the form of the branching factor of an associated tree [10]. Using these concepts, we derive an overall convergence rate of the algorithm to the global infinite-horizon optimum as a function of computation, measured by the number of transitions simulated.

A limitation of this first OPC method is that it requires the Lipschitz constant. In practice the constant is difficult to find so it must be treated as a tuning parameter, which is easy to overestimate (making the algorithm conservative) or underestimate (invalidating the guarantees). So we also propose a second algorithm that expands all potentially optimistic boxes, using only the knowledge that boxes that have been split more times have smaller diameters. This algorithm is called *simultaneous OPC (SOPC)*. We analyze SOPC and show that it has nearly the same convergence rate as OPC, even though it does not need to know the value function smoothness. SOPC relies on a different tuning parameter than the Lipschitz constant, which can however be tuned much more robustly. Simulation results show that SOPC outperforms OPC, and is also better than competing continuous-action planners and baseline dynamic programming and MPC solutions. We provide real-time control results with SOPC.

In contrast to much of the work in nonlinear MPC [7], which uses a fixed finite horizon, OPC and SOPC directly explore the space of infinite-horizon solutions, and therefore our near-optimality bounds and convergence rates are with respect to the global, infinite-horizon optimum. E.g. the closest work to ours is the optimistic MPC method of [21], which only works for small fixed control horizons (and max-plus systems). OPC and SOPC instead adaptively increase the horizon as much as the computation allows. Furthermore, typical MPC methods are derivative-based, while our methods only rely on Lipschitz values, so at the cost of more computation, they can handle dynamics and rewards that are nondifferentiable at some discrete points (on a set of measure zero).

In planning, several other optimistic methods have been proposed for continuous actions, but without an analysis; to our knowledge OPC and SOPC are the first to guarantee a convergence rate. Lipschitz planning (LP) [9] uses a similar upper bound but lacks the insight on the impact γ^k, so it uses a heuristic rule to choose which dimension to split. Our earlier method called simultaneous optimistic optimization for planning (SOOP) [3] is similar to SOPC in that it expands many boxes at once, but uses a heuristic for selecting these sets, which turns out to be worse in our simulations. Other continuous-action planners only optimize over finite horizons, e.g. HOOT [15] or sequential planning [9].

Our new planners apply the *principle* of optimistic optimization (OO) [17] to control, while the *analysis* of OO does not work because its assumptions are not satisfied for infinite-horizon continuous-input problems. Thus, we must provide novel analysis adapted to this setting. The present paper is an extended and revised version of [5], and the material on OPC largely originates in that paper. The major novelty compared to [5] is the SOPC method, with almost as good analytical guarantees and much better practical performance than OPC. Even for OPC, we provide here extra insight that due to space limits was not available in [5]. The simulations are extended and the real-time results are new.

Next, Sec. 2 formalizes the problem and Sec. 3 describes OPC and SOPC. Sec. 4 analyzes the two algorithms, while Sec. 5 provides numerical results. Sec. 6 concludes the paper. Supplementary material is available at `http://busoniu.net/files/papers/sopc_suppl.pdf`.

2 Problem statement

We consider an optimal control problem for a discrete-time nonlinear system $x_{k+1} = f(x_k, u_k)$, where $x \in X \subseteq \mathbb{R}^p$, $u \in U$, and U will be described in our main assumption below. A function $\rho : X \times U \to \mathbb{R}$ assigns a numerical reward $r_k = \rho(x_k, u_k)$ to each state-action pair. Under a fixed initial state x_0, define an infinitely-long sequence of actions $\boldsymbol{u}_\infty = (u_0, u_1, \dots)$ and its infinite-horizon discounted value:

$$v(\boldsymbol{u}_\infty) = \sum_{k=0}^{\infty} \gamma^k \rho(x_k, u_k) \tag{1}$$

where $\gamma \in (0,1)$ is the discount factor, and $x_{k+1} = f(x_k, u_k)\ \forall k \geq 0$. The objective is to find (a near-optimal approximation of) the optimal value $v^* := \sup_{\boldsymbol{u}_\infty} v(\boldsymbol{u}_\infty)$ and an action sequence that achieves this (near-optimal) value. Very general conditions that ensure the existence of optimal sequences are provided e.g. by [2].

We impose some assumptions that allow us to derive efficient algorithms.

Assumption 1 *The following conditions hold.*

(i) The rewards are bounded in $[0, 1]$.
(ii) The action is a real scalar, bounded in the unit interval, so that $U = [0, 1]$.

The main role of reward boundedness 1(i), together with discounting, is to ensure that for any sequence the values in (1) are bounded to $[0, \frac{1}{1-\gamma}]$. Our planning algorithms and analysis rely on this property. Note that many other works in control use discounting, e.g. [6,11]. Bounded costs are typical in AI methods for optimal control, such as the planning class in our focus [14] and reinforcement learning [20]. One way to achieve boundedness is by saturating a possibly unbounded original reward function, which changes the optimal solution but is often sufficient in practice. Another example is when physical limitations in the system are modeled by saturating the states and actions, from which a reward bound follows.

The scalar action from Assumption 1(ii) could in principle be generalized to multiple dimensions; however, computation would grow very fast with action dimensionality, so in practice this will not work for more than a few dimensions. (In the supplementary material, we briefly explain and empirically test such an extension for two dimensions.) The compact nature of U is fundamental, since our algorithm numerically refines this action space. In both Assumptions 1(i) and 1(ii), the unit interval is taken only for convenience, and can be achieved by rescaling any bounded interval.

A crucial requirement is a Lipschitz property of v with respect to its argument $\boldsymbol{u}_\infty$, "one-sided" around optimal sequences.

Assumption 2 *There exists $L_v > 0$ so that for any optimal sequence $\boldsymbol{u}^*_\infty$ and any other sequence $\boldsymbol{u}_\infty \in U^\infty$:*

$$v(\boldsymbol{u}^*_\infty) - v(\boldsymbol{u}_\infty) \leq L_v \sum_{k=0}^{\infty} \gamma^k \left| u^*_k - u_k \right| \qquad (2)$$

Intuitively, this means that the discounted nature of the values should be reflected also in the importance of the control actions along time: later actions should impact the value less than earlier ones. At the cost of some restrictiveness (for instance, constraints imposed via penalty functions in the rewards may not lead to Lipschitz values), Assumptions 1 and 2 will allow us to derive a numerical method with global near-optimality and convergence rate guarantees, which is still very general in that e.g. it does not impose any specific mathematical form for v.

Our development only relies on the general property (2). It will however be instructive to examine some sufficient conditions for this property in a particular case. (Proofs of all results are given in the supplementary material.)

Lemma 3 (Lemma 2 of [5]) *Assumption 2 is ensured with $L_v = \frac{L_\rho}{1-\gamma L_f}$ by the following two conditions:*

(i) The dynamics and rewards are Lipschitz, i.e. $\exists L_f, L_\rho$ so that $\forall x, x' \in X, u, u' \in U$:

$$\|f(x,u) - f(x',u')\| \leq L_f(\|x - x'\| + |u - u'|)$$
$$|\rho(x,u) - \rho(x',u')| \leq L_\rho(\|x - x'\| + |u - u'|)$$

where $\|\cdot\|$ is an appropriately chosen norm.

(ii) The dynamics satisfy $\gamma L_f < 1$.

The Lipschitz conditions 3(i) must hold over the entire domains of f and ρ. Some dynamics and reward functions would not be globally Lipschitz for unbounded states, in which case a bounded X is needed. In practice this corresponds again to state saturation limits, as exemplified in our discussion of Assumption 1 above. Note that unlike in typical derivative-based MPC techniques [7], condition (i) allows f or ρ to be nondifferentiable, but only on a set of measure zero (to preserve Lipschitz continuity). Condition 3(ii) means that the dynamics need not be strictly contractive on their own, but should become so when combined with a shrink rate equal to the discount factor γ.

Example 4 *Linear dynamics and quadratic cost.* Consider the familiar optimal control problem with linear dynamics $x_{k+1} = Ax_k + Bw_k$, scalar inputs w, and discounted quadratic costs $\sum_{k=0}^{\infty} \gamma^k (x_k^\top Q x_k + R w_k^2)$ [1]. Due to physical limits it is known that $-\bar{x} \leq x \leq \bar{x}$ and $-\bar{w} \leq w \leq \bar{w}$, where vector inequalities hold elementwise. To place the problem in our framework, w will be represented by unit actions $u \in [0,1]$ via $w = 2\bar{w}(u - 0.5)$. The system is then modelled by the saturated dynamics: $f(x,u) = \mathrm{sat}(Ax + 2B\bar{w}(u - 0.5), -\bar{x}, \bar{x})$. Minimizing the quadratic costs is equivalent to maximizing the rewards: $\rho(x,u) = 1 - (x^\top Q x + 4R\bar{w}^2(u-0.5)^2)/\bar{r}$, where $\bar{r} = \bar{x}^\top Q \bar{x} + R\bar{w}^2$ is a bound on the cost. So, $\rho(x,u) \in [0,1]$ and Assumption 1 is satisfied.

Further, using the Euclidean norm it is not difficult to prove that f is Lipschitz with $L_f = \max\{\|A\|, 2\bar{w}\|B\|\}$, and ρ is locally Lipschitz over the bounded domains of x and u with $L_\rho = \frac{1}{\bar{r}} \max\left\{2\left\|\bar{x}^\top Q\right\|, 8R\bar{w}^2\right\}$, so Assumption 3(i) is satisfied. To check Assumption 3(ii), numerical values are needed, so take a standard DC motor model with states: angle α and velocity $\dot{\alpha}$, voltage input w, inertia $J = 10^{-4}\,\mathrm{Nm}^2$, torque constant $K = 0.02\,\mathrm{Nm/A}$, resistance $R = 10\,\Omega$, $L = 0.01\,\mathrm{H}$, $b = 3 \cdot 10^{-6}\,\mathrm{Nms/rad}$, and sampling time $0.005\,\mathrm{s}$. With $\bar{w} = 5\,\mathrm{V}$, we get $L_f \approx 1.25$ so any value γ smaller than 0.8 works, thereby guaranteeing Assumption 2. □

3 Planning algorithms for continuous actions

We apply the principles of OO [17] to maximize the objective function v over the space U^∞ of infinitely long sequences. The idea is to iteratively split the search space into smaller subsets, where at each iteration the set to split further is chosen optimistically, so that it (potentially) maximizes an upper bound on the optimal value.

To derive the splitting procedure, U^∞ is represented by an infinite dimensional hyperrectangle, in which each dimension k is the action at step k. This hyperrectangle is iteratively split into smaller hyperrectangles, like in [9,3], each of which gets a unique index i. In the sequel we will prefer the term "dimension" for k, which emphasizes the geometric structure of the search space, and the short form "box" for hyperrectangle. A box is denoted $\mathcal{U}_i \subseteq U^\infty$ and is the cartesian product of a sequence of intervals $(\mu_{i,0}, \ldots, \mu_{i,K_i-1}, U, U, \ldots)$ where $\mu_{i,k} \subseteq U$ and $K_i - 1$ is the largest discretized dimension; for all further dimensions $\mu_{i,k} = U$. A box is refined by splitting into $M > 1$ pieces the interval of some dimension

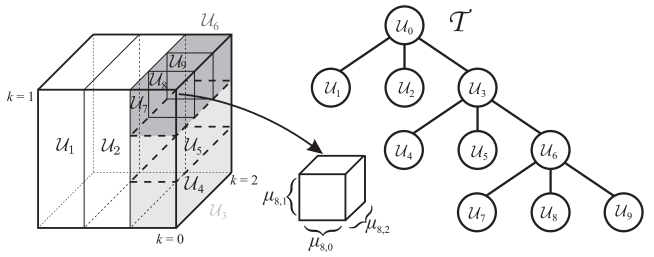

Fig. 1. Left: Example refinement of U^∞ after 3 splits, for $M = 3$. Dimensions 4 and higher are left out of the figure. Boxes that have already been split ($\mathcal{U}_3$,$\mathcal{U}_6$) are shown in shades of gray. Right: Corresponding tree.

$k \le K_i$, which corresponds to discretizing u_k more finely. Define $d_{i,k}$ to be the length of the interval $\mu_{i,k}$ in box $\mathcal{U}_i$, and $u_{i,k}$ a sample action taken somewhere in this interval (e.g., at the center). For each box, the sequence of rewards $r_{i,k}$ obtained by applying the (finite) sequence $u_{i,k}$ from x_0 is computed by simulating the system.

To exemplify, see Fig. 1 and consider e.g. the dark gray box $\mathcal{U}_6$, which consists of the sequence of intervals $(\mu_{6,0}, \mu_{6,1}, U, U, \dots) = ([2/3, 1], [2/3, 1], [0, 1], [0, 1], \dots)$ and has a number of discretized dimensions $K_6 = 2$. Box $\mathcal{U}_6$ is split into 3 children boxes along dimension $k = 2$, among which we consider $\mathcal{U}_8$. This child, as well as all its siblings, inherits the intervals for $k = 0$ and 1, so that $\mu_{8,0} = \mu_{6,0} = [2/3, 1]$ and $\mu_{8,1} = \mu_{6,1} = [2/3, 1]$. Since $\mathcal{U}_8$ is the middle child, its interval for $k = 2$ is $\mu_{8,2} = [1/3, 2/3]$. Now K_8 is 3, larger by 1 than K_6, because we have split the first undiscretized dimension of $\mathcal{U}_6$. Note that the order of dimensions to split was chosen arbitrarily in this example; e.g we might have split again $k = 0$ instead of $k = 2$. We will make this order definite later. When sampling is done at the interval centers, the sample action sequence of box 8 is $(5/6, 5/6, 1/2)$.

The collection of boxes will be organized into a tree $\mathcal{T}$ with the root consisting of the entire space, and where each node has M children, one for each of the M boxes resulting from its splitting, see Fig. 1, right. Each node is labeled by the index i of the box, as well as by the box itself, and we will use these notations interchangeably. The depth h of a box i in this tree is therefore equal to the number of splits needed to obtain the box:

$$h = \sum_{k=0}^{\infty} s_i(k) \tag{3}$$

where $s_i : \{0, 1, \dots\} \to \{0, 1, \dots\}$ is a function that gives the number of splits along dimension k, so $s_i(k) = 0$ when $k \ge K_i$. Note that the root has depth 0, and that $d_{i,k} = M^{-s_i(k)}$. Also, a given box may be obtained along multiple paths along the tree, but for simplicity the algorithm does not make use of this information, and keeps all the copies. In a practical implementation it is of course advisable to merge them. Denote the leaves of $\mathcal{T}$ by $\mathcal{L}$; at any iteration, at least one leaf with the largest upper bound on the optimal value must be refined.

Our next goal is to compute this upper bound. Recall that the information available for a box i includes the sample sequence of actions $u_{i,0}, \dots, u_{i,K_i-1}$ and the corresponding rewards $r_{i,0}, \dots, r_{i,K_i-1}$. Define the sample value of box i as follows:

$$v(i) = \sum_{k=0}^{K_i-1} \gamma^k r_{i,k}$$

Consider now only those boxes i that contain an optimal sequence $\boldsymbol{u}^*_\infty$, and define the infinite sequence obtained by appending optimal actions to the sample sequence: $\boldsymbol{u}_{i,\infty} = (u_{i,0}, \dots, u_{i,K_i-1}, u^*_{K_i}, u^*_{K_i+1}, \dots)$. (This sequence will not be constructed by the algorithms, it is only used as an intermediate step in the derivation.) Using Assumption 2, we have:

$$\begin{aligned} v^* &\le v(\boldsymbol{u}_{i,\infty}) + L_v \sum_{k=0}^{K_i-1} \gamma^k \left| u^*_k - u_{i,k} \right| \\ &\le v(i) + \sum_{k=K_i}^{\infty} \gamma^k \cdot 1 + L_v \sum_{k=0}^{K_i-1} \gamma^k d_{i,k} \\ &\le v(i) + \max\{L_v, 1\} \sum_{k=0}^{\infty} \gamma^k d_{i,k} \end{aligned}$$

where the first step exploits the equality of the tails of the two sequences; the second step uses the box lengths along dimensions $k < K_i$ and the fact that the rewards of sequence $\boldsymbol{u}_{i,\infty}$ at steps $k \ge K_i$ are at most 1; and the third step puts the summation together knowing that $d_{i,k} = 1$ for the unexpanded dimensions $k \ge K_i$. In this way, we have eliminated the need to separately handle discretized and undiscretized dimensions, at the expense of some conservativeness due to the maximum in the Lipschitz constant.

Denote $\bar{L}_v = \max\{L_v, 1\}$ and the *diameter* of box i:

$$\delta(i) = \bar{L}_v \sum_{k=0}^{\infty} \gamma^k d_{i,k} = \bar{L}_v \sum_{k=0}^{\infty} \gamma^k M^{-s_i(k)} \tag{4}$$

This is is indeed a true diameter under the metric from the r.h.s. of (2), but with $\bar{L}_v$ instead of L_v. So, finally, we have obtained an upper bound on the optimal value:

$$v^* \le v(i) + \delta(i) =: b(i) \tag{5}$$

for any box that contains an optimal solution, while the diameter represents an uncertainty on this optimal value. We call $b(i)$ the b-value of box i. Note that our approach uses b-values for all the boxes, despite the fact that they are only meaningful for boxes containing optimal values; this is however safe and the algorithms remain correct, as the analysis will show.

So far, the dimension to split for a given box i was not specified. Given (4), the choice is intuitively clear – split a dimension that has maximal contribution to the diameter, so as to minimize the resulting uncertainty:

$$\arg\max_k \bar{L}_v \gamma^k d_{i,k} = \arg\max_k \gamma^k d_{i,k} \tag{6}$$

This procedure is directly related to the Lipschitz property (2), since it weighs each dimension by its discounted impact on the value function. Because the contribution of first undiscretized dimension K_i is γ^{K_i}, larger than the contributions γ^k of all later dimensions $k > K_i$, the maximization will produce at most K_i. So, the method either refines further an already discretized dimension, or starts splitting the first undiscretized dimension. E.g., at the beginning of the algorithm, dimensions will be split in order, then at some point that depends on the discount factor the algorithm will begin returning to earlier dimensions in a rather complicated way, but always keeping the number of splits larger for earlier dimensions. An adaptive discretization procedure results, leading to discrete-time action sequences of increasing length and precision. In particular, the procedure can asymptotically reach arbitrarily close to any continuous-valued sequence of finite length, see also Theorem 6; although it may never *exactly* reach it. In particular, the interval ends will not be exactly reached, which may be kept in mind for bang-bang solutions.

The derivations above hold for any action sample choice and any value of M. In the sequel, we impose some requirements on these choices.

Assumption 5 *The following conditions hold.*

(i) Splitting any box i produces at least one child box j with a greater or equal value, $v(j) \geq v(i)$.
(ii) $M > 1/\gamma$.

A simple strategy satisfying Assumption 5(i) is to take M odd and the action samples $u_{i,k}$ at the centers of the intervals $\mu_{i,k}$. Then, if the split dimension is less than K_i, the middle child j has the same sequence as i so its value is also the same (and the center rewards of the parent box can be directly reused for this child). When the split dimension is K_i, new positive rewards are added to the end of the summation in $v(i)$, so the inequality holds for any child. Assumption 5(ii) is trivial, being already satisfied by $M \geq 3$ when $\gamma > 1/3$; most problems of practical interest require larger discount factors.

Our first algorithm requires knowledge of the Lipschitz constant $\bar{L}_v$, so it can explicitly compute b-values (5) and select at each iteration a box that is surely optimistic, i.e. that maximizes these b-values. The procedure starts with the full box U^∞, at the root $i = 0$ of the tree, and proceeds by splitting at each iteration such an optimistic box. The resulting method is called *Optimistic Planning with Continuous actions*, OPC, see Alg. 1. At the end, it returns a sample sequence with the largest value v, $\hat{\boldsymbol{u}} = (u_{i^*,0}, \ldots, u_{i^*,K_{i^*}-1})$.

Algorithm 1 OPC

```
1: input: state $x_0$, model $f$, $\rho$, split factor $M$, budget $n$, Lipschitz constant $\bar{L}_v$
2: initialize $\mathcal{T}$ with root 0 labeled by $U^\infty$
3: while computation $n$ not exhausted do
4:     select box $i^\dagger = \arg\max_{i \in \mathcal{L}} b(i)$
5:     select $k^\dagger = \arg\max_k \gamma^k d_{i^\dagger,k}$
6:     expand $i^\dagger$ along $k^\dagger$: create its $M$ children on $\mathcal{T}$
7: output sequence $\hat{\boldsymbol{u}}$ of box $i^* = \arg\max_{i \in \mathcal{L}} v(i)$
```

Algorithm 2 SOPC

```
1: input: state $x_0$, model $f$, $\rho$, split factor $M$, budget $n$, function $h_{\max}$
2: initialize $\mathcal{T}$ with root 0 labeled by $U^\infty$
3: loop
4:     $h$ = smallest depth with unexpanded nodes
5:     if $h > h_{\max}(n)$, stop (go to line 13)
6:     while $h \leq h_{\max}(n)$ do
7:         select box $i^\dagger = \arg\max_{i \in \mathcal{L}_h} v(i)$
8:         select $k^\dagger = \arg\max_k \gamma^k d_{i^\dagger,k}$
9:         expand $i^\dagger$ along $k^\dagger$, updating $\mathcal{T}$
10:        update $n_s$
11:        if $n_s \geq n$, stop (go to line 13)
12:        $h = h + 1$
13: output sequence $\hat{\boldsymbol{u}}$ of box $i^* = \arg\max_{i \in \mathcal{L}} v(i)$
```

The following remarks apply both to OPC and the next algorithm. Computation is measured by the number n of evaluations of the model, i.e. of the pair f, ρ, since for a nonlinear system simulating f is often expensive. At worst, the cost of expanding one box is $MK_{i^\dagger}$ evaluations. More specifically, with the center sampling strategy explained above and by reusing samples, the cost is M model calls when $k^\dagger = K_{i^\dagger}$, and $(M-1)(K_{i^\dagger} - k^\dagger)$ otherwise. On reaching the budget limit the algorithm may either be allowed to expand its last box, or immediately stopped while rolling back the changes made at the interrupted step.

The second algorithm does not use b-values, so it does not require knowledge of $\bar{L}_v$. The price paid is expanding several nodes per iteration, but as the analysis and experiments will show, this does not greatly impact either the bounds or the empirical performance. Specifically, at each iteration, one node maximizing v is expanded at each depth h (recall that depths are given by (3)). This is the best guess at an optimistic node without knowing the diameters. To avoid the expansions continuing indefinitely, a maximum depth $h_{\max} : \{0, 1, \ldots\} \to [0, \infty)$ is enforced, which takes as argument the budget n. Function $h_{\max}$ is a parameter of the algorithm. If it grows quickly with n then it allows deep searches, and if it grows slowly, it favors exploring broad trees. Since it expands several nodes at one iteration, the algorithm is called *Simultaneous OPC* or SOPC, see Alg. 2. The early stopping condition in line 5 should never be activated in normal use; if it does, $h_{\max}$ should be increased. We discuss more on choosing $h_{\max}$ (and $\bar{L}_v$, M) once we have the benefit of the analysis, at the end of Sec. 4.

While OPC has greater limitations than SOPC, it is important as a stepping stone to understand SOPC and the analysis line. SOPC extends the idea behind simultaneous OO (SOO) [17] to planning. Note that in SOO a certain depth may be skipped during the expansion loop, if the leaves at that depth have smaller values than those at higher depths, which means they cannot be optimistic. In SOPC, as soon as a depth with unexpanded nodes is found, the structure imposed by Assumption 5(i) ensures that each subsequent depth will have at least as large a maximum value, so a contiguous range of depths is always expanded.

While both OPC and SOPC apply the main *ideas* behind OO, the *guarantees* of OO cannot be directly applied, since the boxes obtained do not satisfy certain geometric properties required by these guarantees [17]. Thus, we need to provide novel analysis, adapted to the setting of optimal control, which we proceed to do next.

4 Analysis

The main objective is to characterize the near-optimality of OPC and SOPC as a function of the computational budget n invested. We start in Sec. 4.1 with preliminary results required for both algorithms. Since near-optimality is driven by box diameters at certain depths, we first characterize these diameters in Theorem 6. This is a major, nontrivial step due to the infinite dimensionality of each box. Then, we define a measure of problem complexity, in the form of an effective branching factor of a subtree containing near-optimal nodes (Definition 7).

We will focus on OPC in Sec. 4.2, where we initially provide an a posteriori guarantee, standard for optimistic algorithms, which shows that the sub-optimality is at most the smallest diameter among expanded boxes (Lemma 8). Using the preliminary results, we then put the computational budget in relation to this diameter, leading to an a priori near-optimality in Theorem 9.

Sec. 4.3 moves to SOPC. Here, the depth of the node whose diameter dictates near-optimality is more involved to characterize than in OPC, and is analyzed in Lemma 10 using the branching factor. Finally, like for OPC, an a priori bound is obtained in Theorem 11.

4.1 Preliminaries

Theorem 6 (Proposition 4 of [5]) *For some $c > 0$ and each box i at any depth h, $\delta(i) \le c\sqrt{2h(\tau-1)}\gamma^{\sqrt{2h\frac{\tau-1}{\tau^2}}} =: \delta_h$, where $\tau = \left\lceil \frac{\log M}{\log 1/\gamma} \right\rceil$.*

Since $\gamma < 1$, the term $\gamma^{\sqrt{2h\frac{\tau-1}{\tau^2}}}$ asymptotically dominates $\sqrt{2h(\tau-1)}$, and makes the diameter converge to zero; in asymptotic notation [1], $\delta_h = \tilde{O}\left(\gamma^{\sqrt{2h\frac{\tau-1}{\tau^2}}}\right)$. Convergence is exponential not directly in the depth h, but in its square root modulated by $2(\tau-1)/\tau^2$.

Both planning algorithms may explore at h the following set of near-optimal nodes:

$$\mathcal{T}_h^* = \{ i \text{ at } h \,|\, v(i) + \delta_h \ge v^* \}$$

Define also $\mathcal{T}^* = \bigcup_{h\ge 0} \mathcal{T}_h^*$, which will generally be *smaller* than the full tree. Rewriting the condition as $v^* - v(i) \le \delta_h$, we see that for each node in $\mathcal{T}_h^*$, given the available value information $v(i)$ and the uncertainty δ_h, the algorithm is still unsure whether the node contains an optimal solution, so it may need to refine this node further. Next, we characterize of the size of $\mathcal{T}^*$.

Definition 7 *The asymptotic branching factor is the smallest m so that $\exists C \ge 1$ for which $|\mathcal{T}_h^*| \le Cm^h, \forall h$, where $|\cdot|$ denotes set cardinality.*

This branching factor is a measure of the complexity of the planning problem. It is similar to the asymptotic branching factor in OPD [10] and plays a related role to other measures of problem complexity used to characterize optimistic methods, see e.g. [18]. However, its meaning is different in our continuous-action optimal control problem. Note that m may be noninteger but lies in the interval $[1, M]$ – since there is at least one node in $\mathcal{T}_h^*$, the one containing the optimal solution; and at most M^d, the entire set of nodes. A smaller value of m corresponds to a simpler problem, with the easiest problems having $m = 1$, e.g. when a constant number of optimal paths must be explored.

4.2 OPC performance

We start with a basic property of OPC, and with a simple bound that can be directly computed by the algorithm, and therefore provides an a posteriori guarantee.

Lemma 8 (Proposition 3 of [5]) *OPC only expands boxes $i \in \mathcal{T}^*$. The sub-optimality $v^* - v(\hat{\boldsymbol{u}})$ of the returned sequence is at most $\delta_{\min}$, the smallest diameter among all expanded boxes.*

These properties are standard for optimistic algorithms. Next, we are interested in a stronger, a priori guarantee, which requires novel derivations adapted to OPC. We exploit the diameter bound of Theorem 6 together with the branching factor of Definition 7 to obtain a bound directly in terms of the budget n.

[1] Let $g, h : (0, \infty) \to \mathbb{R}$. Statement $f(t) = \tilde{O}(g(t))$ means that $\exists a > 0, b \ge 0$ so that $f(t) \le a(\log g(t))^b g(t), \forall t > 0$. When the statement is made for large t, the inequality must hold for $\forall t \ge t_0$ where $t_0 > 0$.

Theorem 9 (Theorem 6 of [5]) *For large budgets n:*

- *When $m > 1$, we have: $v^* - v(\hat{\boldsymbol{u}}) = \tilde{O}\left(\gamma\sqrt{\frac{2(\tau-1)\log n}{\tau^2 \log m}}\right)$.*
- *When $m = 1$, $v^* - v(\hat{\boldsymbol{u}}) = \tilde{O}(\gamma^{n^{1/4}a})$ where $a = \sqrt{\frac{2(\tau-1)}{\tau^2\sqrt{MC}}}$ and C is the constant from Definition 7.*

Thus, the convergence rate of OPC is modulated by the problem complexity as expressed by branching factor m. The smaller m, the easier the problem and the faster the bound reduces with increasing n. In particular, when $m = 0$ the bound is exponential in $n^{1/4}$ – faster than in the general case, where it is exponential in $\sqrt{\log n}$.

4.3 SOPC performance

Similarly to Lemma 8 for OPC, we first establish a lower bound for the depth reached by SOPC after spending a certain computation budget n.

Lemma 10 *Define $h(n)$ as the smallest value h for which the following inequality holds:*

$$CMh_{\max}^2(n)\sum_{h'=0}^{h} m^{h'} \geq n \tag{7}$$

Then, SOPC expands a node containing the optimal solution at depth $\underline{h} = \min\{h(n), h_{\max}(n)\}$, and the sequence returned is $\delta_{\underline{h}}$-optimal.

Lemma 10 adapts the SOO depth bound from [18], in order to take into account the varying number of model calls to create a child box, while in SOO a child only needs one call to the objective function.

Next, we combine the depth bound with the diameter from Theorem 6 to establish the final guarantee for SOPC, directly linking performance to computation.

Theorem 11 *Consider the sequence $\hat{\boldsymbol{u}}$ returned by SOPC. For large n:*

- *When $m > 1$, take $h_{\max}(n) = n^{\varepsilon}$ with $\varepsilon \in (0, 0.5)$. Then, $v^* - v(\hat{\boldsymbol{u}}) = \tilde{O}\left(\gamma\sqrt{\frac{2(\tau-1)(1-2\varepsilon)\log n}{\tau^2 \log m}}\right)$.*
- *When $m = 1$, take $h_{\max}(n) = n^{1/3}$. Then, $v^* - v(\hat{\boldsymbol{u}}) = \tilde{O}(\gamma^{n^{1/6}b'})$ where $b' = \sqrt{\frac{2(\tau-1)}{\tau^2}\min\left\{\frac{1}{CM}, 1\right\}}$ and C is the constant from Definition 7.*

Thus, by transplanting the strengths of SOO from optimization to control, we get an algorithm which has nearly the same convergence rate as OPC when $m > 1$, since the extra term $1 - 2\varepsilon$ can be made small by taking small ε. This is because the extra polynomial cost of expanding complete paths in the tree is small compared to the overall exponential number of boxes. When $m = 1$, convergence is exponential in $n^{1/6}$ instead of $n^{1/3}$ – still fast, although slower than OPC.

Regarding the tuning of $h_{\max}(n)$, due to Theorem 11 it should be taken n^{ε} with $\varepsilon \in (0, 0.5)$. A first idea is to take it $n^{1/3}$, in the hope that the problem is simple and accepting a possible slower convergence rate if it turns out to be more complicated. Comparing to OPC, where another parameter must be tuned – the Lipschitz constant $\bar{L}_v$ – $h_{\max}(n)$ is much more robust. Indeed, choosing a bad value for ε only reducess the convergence rate of SOPC, while the algorithm remains valid. On the other hand, if $\bar{L}_v$ is underestimated, the explicit b-values used by OPC become invalid. Beyond this, the value of $\bar{L}_v$ is problem-dependent, where the danger of underestimation must be balanced with the effects of overestimation: uninformative b-values slowing down the algorithm. Furthermore, since the SOPC analysis holds without knowing $\bar{L}_v$, we can replace its most favorable value while keeping the analysis valid. So the algorithm can be thought as *automatically adapting to the unknown smoothness of the dynamics and reward function*, thus becoming (nearly) as good as OPC with an optimally tuned Lipschitz constant. See the upcoming experiments for an empirical study of the effect of these parameters.

The last parameter to be chosen is the number of splits M. A larger M reduces the term $\frac{\tau-1}{\tau^2}$ in the bounds, improving them, but it may also lead to a larger branching factor m, with the opposite effect. Overall we estimate the latter danger is more significant, so we suggest $M = 3$, the smallest value satisfying Assumption 5(i).

5 Experiments with the rotational pendulum

Our example is the Quanser rotational pendulum, see Fig. 2(left), which consists of a heavy rod sitting on an unactuated rotational joint at the end of an intermediate, horizontal link actuated through a motor. The problem has four state variables, $x = [\theta, \dot{\theta}, \alpha, \dot{\alpha}]^\top$, $\theta \in [-\pi, \pi)\,\text{rad}$ is the angle of the horizontal link (zero pointing forward), where $\alpha \in [-\pi, \pi)\,\text{rad}$ is the angle of the pendulum (zero pointing up), and $\dot{\theta}, \dot{\alpha} \in [-100, 100]\,\text{rad/s}$ are the angular velocities, with these limits enforced by saturation. The input voltage u lies in $[-9, 9]\,\text{V}$, see [5] for details about the dynamics. The goal is to reach the zero equilibrium, but the system is underactuated so the pendulum must first be swung back and forth to accumulate energy, which means that long trajectories must be found. The problem is therefore challenging for planning methods. The unnormalized reward function is the usual quadratic one: $\tilde{\rho}(x, u) = -x^\top \text{diag}[1, 0.005, 1, 0.005]x - 0.05u^2$, see Example 4, where the weights are chosen to prioritize angle stabilization, followed by the angular velocity and the control effort (keeping in mind the velocity range is large). The sampling time is chosen $0.05\,\text{s}$, and $\gamma = 0.85$.

We begin by pitting SOPC and OPC against two heuristic planners with continuous actions. We use OPC with a tighter b-value that exploits the formulas in the proof of Lemma 3, together with the knowledge that rewards are at most 1 and that action samples are always at the centers of the intervals:

$$\sum_{k=0}^{K_i-1} \gamma^k \min\{1, r_{i,k} + L_\rho \sum_{j=0}^{k} L_f^{k-j} \frac{d_{i,j}}{2}\} + \frac{\gamma^{K_i}}{1-\gamma} \quad (8)$$

This form would have been unwieldy to use in the analysis, but in practice the algorithm behaves better (and conservatively satisfies the analysis). The first competing algorithm is Lipschitz planning (LP) [9] in a variant that uses bound (8) but selects dimensions using its own rule, different from OPC and SOPC. The second is simultaneous optimistic optimization for plannning (SOOP) [3], which like SOPC expands multiple sets at each iteration, selecting them with a heuristic that does not ensure performance guarantees; while dimension selection uses a criterion similar to (6), but with a tuning parameter α replacing γ.

For each simulation experiment, each algorithm is run in receding horizon for 50 steps from initial state $\alpha = -\pi$, $\dot{\alpha} = \theta = \dot{\theta} = 0$, the resulting sequence of rewards r_{k+1} is found, and the return $\sum_{k=0}^{49} \gamma^k r_{k+1}$ is reported. In a first experiment, we study the effects of each planner's tuning parameters, leaving the comparison between different planners for a second experiment. We take a large budget of $n = 2000$ so that the planners can find a near-optimal solution. In SOPC, we tune the power ε in the maximum-depth function $h_{\max} = n^\varepsilon$, which the analysis indicates is a good form, taking $\varepsilon \in \{1/4, 1/3, 0.4, 0.45\}$. In OPC and LP, we tune equal Lipschitz constants $L_\rho = L_f =: L$, varying L in the set $\{0.3, 0.5, 0.7, 0.9, 1, 1.1, 1.3, 1.5, 2\}$. For SOOP, α is varied in $\{0.5, 0.7, 0.8, 0.85, 0.9, 0.95\}$. Fig. 2(right) shows the results.

For SOPC, $\varepsilon = 0.45$ works best, for which $h_{\max}$ grows relatively fast with n, indicating that deep trees are preferred in this problem. Note that the analysis predicts good values of ε are less than 1/3, likely because it focuses on the behavior for large n. The larger ε is not surprising in practice, since it promotes exploring longer trajectories for smaller budgets, and long trajectories are needed to perform the swingup. In both OPC and LP, Lipschitz constants around 1 perform well (with the optimum being 1.1 for OPC and 1 for LP), while for SOOP $\alpha = 0.8$ (close to $\gamma = 0.85$) is best. Concerning sensitivity with respect to parameter variations, SOPC and SOOP have similar and slowly varying performance for all values of their tuning parameters; while OPC and LP are quite sensitive to the Lipschitz constants. Overall, it seems Lipschitz constants are difficult to tune and algorithms that do not use them are more reliable.

We run the planners again with the best-performing parameters above, while gradually increasing the budget $n = 100, 200, 500, 1000, 1500, 2000, 3000, 5000, 10000$. Fig. 3 shows the results (VI means value iteration, discussed later). LP works poorly and unpredictably in this problem, confirming earlier results on a range of problems in [3]. Other algorithms generally increase their performance reliably with n. OPC is the next best method, possibly because its dimension selection criterion (6) is better founded than in LP. Next comes SOOP and finally SOPC, the best algorithm in the experiment, showing the benefit of replacing the heuristics in SOOP with good box selection rules. SOPC obtains a significant improvement over tuned OPC, although the analysis 'only' predicts it will be as good as OPC with an optimally tuned Lipschitz constant. This is because the tuning was done for the overall trajectory, while the best Lipschitz constant may be different at each step, and SOPC is able to adapt to these changes.

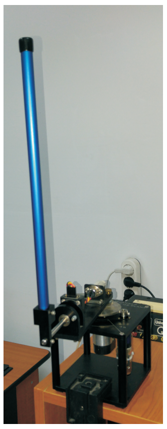
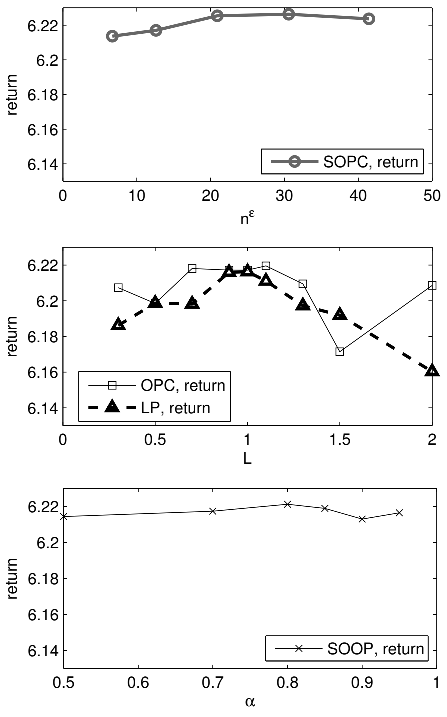

Fig. 2. Left: Rotational pendulum. Right: Planner performance as a function of the tuning parameters; note that for SOPC the graph directly places $h_{\max}(n) = n^\varepsilon$ on the horizontal axis.

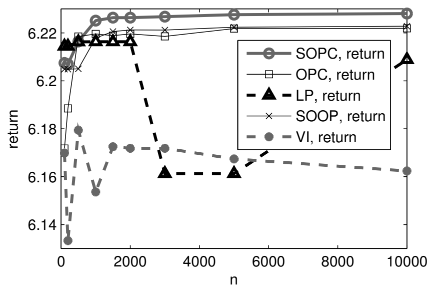

Fig. 3. Algorithm performance for varying budgets.

Note that the differences in return values from Fig. 3 correspond to real differences in control performance; for example, the level reached by OPC and SOOP means

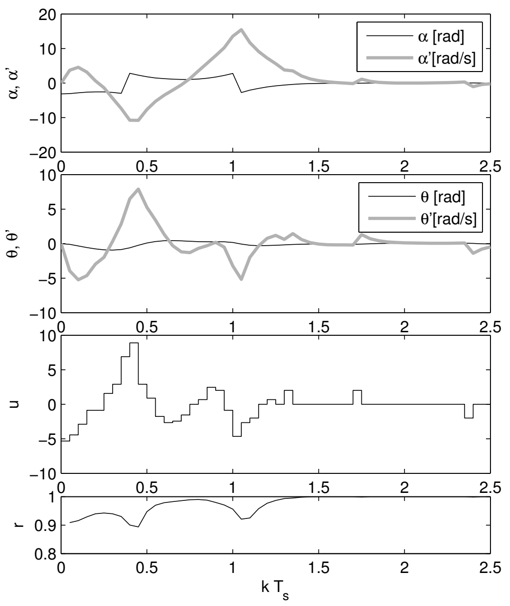

Fig. 4. Rotational pendulum trajectory with SOPC.

that after swinging up the pendulum, it is not accurately stabilized, but "dropped" and rotated again. In contrast, the higher level reached by SOPC means the pendulum is stabilized well. This good behavior is illustrated for $n = 5000$ in Fig. 4. Because the budget is finite, the adaptive discretization of the actions produces an *approximation* of the continuous-valued optimal controls, with a limited resolution (e.g. keeping the system near the unstable equilibrium still requires adjustments). Nevertheless, the figure shows that SOPC uses many different discrete levels in its effort to optimize the rewards.

Regarding the execution time of the planners, it is directly dictated by (and so roughly linear in) the budget n of simulations allowed, see also the detailed results in the supplementary material.

Next, we compare to a classical offline solution: value iteration with an interpolation grid over the state space and a discretized action grid [4]. For fairness, we choose a number of points on each state and action axis equal to the smallest N so that $50n \leq N^5$, so value iteration is allowed at least as many samples as the planners use across all 50 steps. These points are placed equidistantly along the domains of the variables. The returns obtained by controlling the system with the resulting state feedback policy are shown alongside the planners in Fig. 3 (aligned with the values of n for clarity, even though value iteration may in fact use more samples). They are worse than all planners except LP, which is because the algorithm suffers from the curse of dimensionality: since it takes N^5 points to cover the state-action space, the resulting values of N (at most 14) are too small to represent a good solution.

Further, we illustrate the resilience of SOPC to non-differentiable, but still Lipschitz dynamics and rewards.

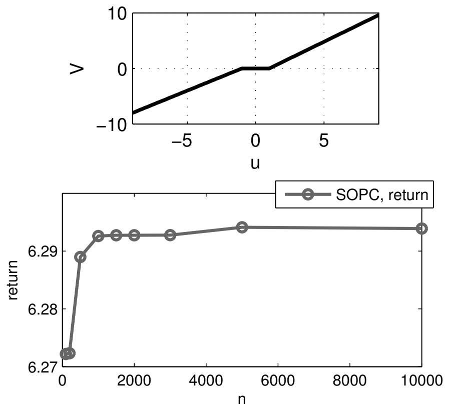

Fig. 5. Deadzone (top) and returns obtained by SOPC (bottom). The deadzone is over $u \in [-1, 1]$, and the slopes are 1 and 1.2 to its left and right, respectively.

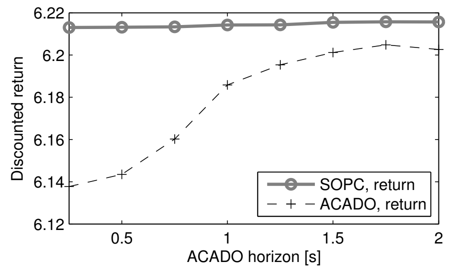

Fig. 6. Comparison with ACADO.

Typical actuator nonlinearities include deadzones and different gains when changing the input sign, and we modify the pendulum dynamics to include both, see Fig. 5, top. We replace in the rewards the input penalty u^2 by $|u|$. Fig. 5, bottom shows the returns of SOPC; all budgets starting from 3000 solve the problem with one swing. Note the returns are not directly comparable with the other experiments.

Since our unoptimized Matlab implementation would not work in real time (e.g. for $n \geq 200$ the execution time already exceeds $T_s = 0.05$ s), we also implement SOPC in C++. We run a final comparison with nonlinear MPC, using the C++ ACADO toolkit [8]. At each step, a finite-horizon optimal control problem is constructed using the negative of the reward function above, without discounting and with the control bounds included as constraints. Receding-horizon MPC is run for horizons $0.25, 0.5, \dots, 2$ s, with the default settings for the optimizer. For each horizon, the average execution time of ACADO is measured, and then the C++ variant of SOPC is run allowing it the same computation time. Looking at the discounted returns in Fig. 6, SOPC is better in all experiments; although it should be noted that when evaluated by the *un*discounted return, ACADO becomes better for large horizons. This is a reflection of the different optimal control problems solved by the two algorithms, each of them being better for the type of cost it targets.

We close the evaluation with a real-time control experiment. Here, while the current action u_k is being applied to the system on an I/O thread, SOPC is already run on another thread from the predicted state at $k+1$, so that a new action is ready by that time. The computer uses an Intel Xeon E5-1620 CPU at 3.6GHz (quad-core but we use two threads as explained above), with 16GB RAM. The C++ implementation is used and the budget is set to 5000, large enough so that most of the sampling period is exploited to plan, but still ensuring the algorithm terminates within 0.05 s. The voltage limit is increased to 14V. A video illustrating the results is at `http://rocon.utcluj.ro/files/rotpend_sopc.avi`.

6 Conclusions and future work

We introduced two optimal control algorithms for nonlinear systems with scalar inputs: optimistic planning for continuous actions (OPC), and simultaneous OPC. Given a Lipschitz-continuous value function, OPC and SOPC are the first algorithms for action sequence search that provide convergence rates to the global infinite-horizon optimum for general, nondifferentiable dynamics and rewards. This is achieved by a branch-and-bound search that adaptively increases the horizon as more computation is allowed. SOPC obtains nearly the same rate as optimally-tuned OPC, despite not needing to know the Lipschitz constant. This feature turned out to have significant practical advantages, leading to good results in simulations. SOPC was also illustrated for the real-time control of a physical system.

OPC and SOPC are zero-order algorithms, i.e. they rely only on value function samples, without derivative information. While one cannot do better when the value function is indeed nondifferentiable, an important next step is extending the methods to exploit higher-order information *whenever it exists*. Building on the analytical understanding developed here, this extension must preserve the global near-optimality of (S)OPC, but whenever possible should also use derivative information to search more efficiently and reduce computations. Another, longer-term goal is to exploit the near-optimality guarantees in order to show that the solution obtained is also stable in closed loop, by extending the discounted stability framework of [19].

References

[1] D. P. Bertsekas, *Dynamic Programming and Optimal Control*, 4th ed. Athena Scientific, 2012, vol. 2.

[2] D. P. Bertsekas and S. E. Shreve, *Stochastic Optimal Control: The Discrete Time Case.* Academic Press, 1978.

[3] L. Buşoniu, A. Daniels, R. Munos, and R. Babuška, "Optimistic planning for continuous–action deterministic systems," in *2013 IEEE International Symposium on Adaptive Dynamic Programming and Reinforcement Learning (ADPRL-13)*, Singapore, 16–19 April 2013.

[4] L. Buşoniu, D. Ernst, B. De Schutter, and R. Babuška, "Approximate dynamic programming with a fuzzy parameterization," *Automatica*, vol. 46, no. 5, pp. 804–814, 2010.

[5] L. Buşoniu, E. Páll, and R. Munos, "Discounted near-optimal control of general continuous-action nonlinear systems using optimistic planning," in *Proceedings 2016 American Control Conference (ACC-16)*, Boston, US, 6–8 July 2016.

[6] J. Filar, V. Gaitsgory, and A. Haurie, "Control of singularly perturbed hybrid stochastic systems," *IEEE Transactions on Automatic Control*, vol. 46, no. 2, pp. 179–190, 2001.

[7] L. Grüne and J. Pannek, *Nonlinear Model Predictive Control: Theory and Algorithms*, 2nd ed. Springer, 2016.

[8] B. Houska, H. Ferreau, and M. Diehl, "ACADO Toolkit – An Open Source Framework for Automatic Control and Dynamic Optimization," *Optimal Control Applications and Methods*, vol. 32, no. 3, pp. 298–312, 2011.

[9] J.-F. Hren, "Planification optimiste pour systèmes déterministes," Ph.D. dissertation, Lille 1 University - Science and Technology, 2012.

[10] J.-F. Hren and R. Munos, "Optimistic planning of deterministic systems," in *Proceedings 8th European Workshop on Reinforcement Learning (EWRL-08)*, Villeneuve d'Ascq, France, 30 June – 3 July 2008, pp. 151–164.

[11] K. Katsikopoulos and S. Engelbrecht, "Markov decision processes with delays and asynchronous cost collection," *IEEE Transactions on Automatic Control*, vol. 48, no. 4, pp. 568–574, 2003.

[12] D. E. Kirk, *Optimal Control Theory: An Introduction.* Dover Publications, 2004.

[13] L. Kocsis and C. Szepesvári, "Bandit based Monte-Carlo planning," in *Proceedings 17th European Conference on Machine Learning (ECML-06)*, Berlin, Germany, 18–22 September 2006, pp. 282–293.

[14] S. M. La Valle, *Planning Algorithms.* Cambridge University Press, 2006.

[15] C. Mansley, A. Weinstein, and M. L. Littman, "Sample-based planning for continuous action Markov decision processes," in *Proceedings 21st International Conference on Automated Planning and Scheduling*, Freiburg, Germany, 11–16 June 2011, pp. 335–338.

[16] K. Máthé, L. Buşoniu, R. Munos, and B. D. Schutter, "Optimistic planning with a limited number of action switches for near-optimal nonlinear control," in *Proceedings 53nd Conference on Decision and Control (CDC-14)*, Los Angeles, USA, 15–17 December 2014, pp. 3518–3523.

[17] R. Munos, "Optimistic optimization of a deterministic function without the knowledge of its smoothness," in *Advances in Neural Information Processing Systems 24*, J. Shawe-Taylor, R. S. Zemel, P. L. Bartlett, F. C. N. Pereira, and K. Q. Weinberger, Eds., 2011, pp. 783–791.

[18] ——, "From bandits to Monte Carlo tree search: The optimistic principle applied to optimization and planning," *Foundations and Trends in Machine Learning*, vol. 7, no. 1, pp. 1–130, 2014.

[19] R. Postoyan, L. Buşoniu, D. Nešić, and J. Daafouz, "Stability analysis of discrete-time infinite-horizon optimal control with discounted cost," *IEEE Transactions on Automatic Control*, vol. 62, no. 6, pp. 2736–2749, 2017.

[20] Cs. Szepesvári, *Algorithms for Reinforcement Learning.* Morgan & Claypool Publishers, 2010.

[21] J. Xu, T. van den Boom, and B. D. Schutter, "Optimistic optimization for model predictive control of max-plus linear systems," *Automatica*, vol. 74, pp. 16–22, 2016.