Optimistic Planning for Continuous-Action Deterministic Systems

Lucian Buşoniu*, Alexander Daniels†, Rémi Munos‡, Robert Babuška†
*Université de Lorraine, CRAN, UMR 7039 and CNRS, CRAN, UMR 7039, France (lucian@busoniu.net)
†DCSC, Delft University of Technology, the Netherlands (alexanderdaniels87@gmail.com, r.babuska@tudelft.nl)
‡Team SequeL, INRIA Lille-Nord Europe, France (remi.munos@inria.fr)

***Abstract*—We consider the class of online planning algorithms for optimal control, which compared to dynamic programming are relatively unaffected by large state dimensionality. We introduce a novel planning algorithm called SOOP that works for deterministic systems with continuous states and actions. SOOP is the first method to explore the true solution space, consisting of infinite sequences of continuous actions, without requiring knowledge about the smoothness of the system. SOOP can be used parameter-free at the cost of more model calls, but we also propose a more practical variant tuned by a parameter α, which balances finer discretization with longer planning horizons. Experiments on three problems show SOOP reliably ranks among the best algorithms, fully dominating competing methods when the problem requires both long horizons and fine discretization.**

I. Introduction

Optimal control problems arise in numerous areas of technology. They can be modeled as Markov decision processes, in which optimality is measured by a cumulative reward signal that must be maximized (the return). Standard model-based techniques for this problem are called (approximate) dynamic programming [1], [2], and because they search for a global solution, their complexity grows fast with problem dimensionality. Furthermore, to be feasible in practice many of these methods require discretized actions.

This paper is concerned with a different, *online planning* class of techniques, which work in a local fashion by finding actions on demand for each encountered state. We propose a novel planning technique that works for deterministic systems and *continuous* (though still low-dimensional) actions.

The local nature of planning methods reduces their dependence on state dimensionality in comparison to dynamic programming, and allows most methods—including ours—to naturally deal with continuous states. At each step, an explorative search is made through the space of possible action sequences from the current state, after which the best first action found is applied; the process then repeats at the next step. Planning techniques are thus a very general type of model-predictive control. Since computation is limited in the online setting, the search must be efficient, and a good way to achieve this is *optimistic* search, which explores the most promising regions first. Such optimistic planning (OP) algorithms are better developed in the discrete-action case [3]–[7], with Upper Confidence Trees perhaps the most widely known technique [3]. Our method is closer to OP for deterministic [4] and discrete stochastic systems [7].

*L. Buşoniu is also associated with the Department of Automation, Technical University of Cluj-Napoca, Romania.

Several OP methods also exist for continuous actions. HOOT [8] and SP [9, Ch. 5] rely on the principle of Upper Confidence Trees: they explore the space of sequences of a given length (planning horizon) K, optimizing for the kth action the return obtained over subsequent steps. HOLOP[1] [10] optimizes directly the K-horizon return relative to the initial state (at $k = 0$). All three methods are limited by searching for a sequence that is only optimal over horizon K, whereas the control problem is infinite-horizon. In principle, K can be taken sufficiently large, but this will waste computation, and in practice K is a problem-dependent parameter.

The actual space that should be explored is that of infinitely long continuous-action sequences. To our knowledge, the only existing OP algorithm that does this is Lipschitzian planning (LP) [9, Chapter 5]. LP iteratively splits the infinite-dimensional space into hyperboxes of increasing dimensionality, guided by upper bounds on the return of all sequences within a hyperbox. To compute the bounds, LP requires globally Lipschitz dynamics and rewards, with a known Lipschitz constant. However, the system may not be Lipschitz, or even if it is, its smoothness will usually vary across the state-action space. In the latter case, the Lipschitz constant will be conservative, leading to poor performance in smoother regions.

We propose here a method that does not rely on the restrictive assumption of a known, global Lipschitz constant. Simultaneous optimistic optimization (SOO) is exploited, an algorithm that only requires local smoothness around an optimum, without knowing the Lipschitz constant or indeed even the metric [11]. We develop a nontrivial extension of SOO to the optimization of return over infinitely long action sequences, and call the resulting algorithm *SOO for planning (SOOP)*. The idea is to select at every iteration *all* hyperboxes that are potentially optimal for any metric – rather than the box with the largest upper bound in the given metric, like LP. Then for each selected box, a choice is made on the dimension to split further, guided by a tuning parameter (a parameter-free variant that expands all dimensions is also possible but is computationally less efficient).

Compared to SOO, the main novelty introduced by SOOP is a relaxed selection procedure for potentially optimal boxes. This is necessary because (roughly speaking) SOO would require sorting boxes by their diameter in the unknown metric, which is impossible in the planning problem. The relaxation works under a weaker, reasonable assumption on the ordering

[1]The acronyms stand for: hierarchical optimistic optimization applied to trees (HOOT), hierarchical open-loop optimistic planning (HOLOP), and sequential planning (SP).

of diameters. Due to this difference and other particularities of planning, the analysis of SOOP is currently open. However, we expect good performance on the basis of the analysis of SOO, which guarantees convergence to an optimum at the most favorable rate given by any valid metric.

Note that HOOT and HOLOP also work in stochastic problems, whereas SOOP works in deterministic ones.

Next, Section II provides background about the planning problem, SOO, and LP, making some minor improvements to LP along the way. Section III introduces SOOP, and Section IV empirically compares it with LP, with HOLOP, which is selected as a state-of-the-art finite-horizon method, and with OP for deterministic systems [4], selected to provide a discrete-action baseline. Section V concludes the paper.

II. Background

A. Problem setting

We consider discrete-time, deterministic optimal control problems with continuous state spaces X and continuous action spaces U. The system state changes according to $x' = f(x, u)$, where $f : X \times U \to X$ is the transition function, and the quality of transitions is measured by the bounded reward function $r(x, u, x')$, where $r : X \times U \times X$. All the algorithms we consider work locally for a given state of the system, so throughout the development we focus on such a state. Denote this state by x_0, after setting by convention the current time to 0. Keeping in mind that the entire optimization problem depends on x_0, for notation simplicity we keep this dependence implicit in the sequel.

Define a sequence of K actions $\mathbf{u}_K = (u_0, u_1, \dots, u_{K-1}) \in U^K$, and $\mathbf{u}_\infty \in U^\infty$ an infinitely-long action sequence. The space of solutions that our method will explore is U^∞. The discounted value of a sequence $\mathbf{u}_\infty$ is:

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

where $x_{k+1} = f(x_k, u_k)$, and $\gamma \in (0, 1)$ is the discount factor. The optimal value is:

$$v^* = \sup_{\mathbf{u}_\infty} v(\mathbf{u}_\infty) \tag{2}$$

The truncated return of a sequence with finite length K is:

$$R(\mathbf{u}_K) = \sum_{k=0}^{K-1} \gamma^k r(x_k, u_k) \tag{3}$$

The following assumptions are imposed on the problem.

Assumption 1: (i) The action is scalar. (ii) The action space is $U = [0, 1]$. (iii) Rewards are bounded in the interval $[0, 1]$.

Part (i) is for convenience only, as it allows us to introduce the derivations and the algorithm in a simple fashion. Part (ii) allows any closed interval by translation and scaling, but noncompact action spaces are forbidden. Part (iii) is not restrictive for problems that do not have terminal states[2]; in these problems the rewards can be normalized to $[0, 1]$ without changing the optimal solutions.

In the scalar case, U^∞ can be visualized as an infinite dimensional hypercube on which each dimension represents the action space at that step. The goal of continuous-action planning is to explore U^∞ in such a way that after a computational budget is exhausted, a near-optimal action sequence $\mathbf{u}$ is returned. The method we develop explores U^∞ by iteratively splitting it into hyperboxes (boxes, for short). Such a box $\mathcal{U}_i \subseteq U^\infty$ is the cross-product of a sequence of subspaces $(\mu_0^i, \dots, \mu_{K_i-1}^i, U, U, \dots)$ where $\mu_k^i \subseteq U$ and $K_i - 1$ is the deepest discretized dimension; for all further dimensions $\mu_k^i = U$. Thus K_i is the number of discretized dimensions, and might be seen as a "length" of $\mathcal{U}_i$. A box is further explored by trisecting either the subspace of an already discretized dimension, or the space U for the first undiscretized dimension, K_i. Thus trisecting dimension k corresponds to discretizing the action at step k. In Fig. 1 an example exploration of U^∞ is shown.

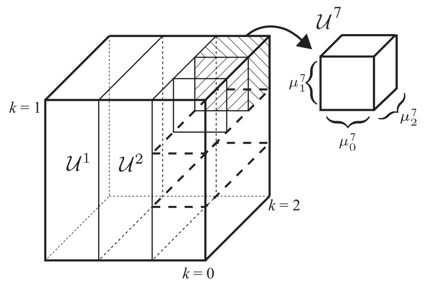

Fig. 1. Example partition of U^∞ after 3 trisections. Dimensions 4 and higher are left out of the figure.

Define δ_k^i to be the size of subspace μ_k^i in box $\mathcal{U}_i$:

$$\delta_k^i = \begin{cases} \max_{u \in \mu_k^i} |u_k^i - u| & \text{for } 0 \le k < K_i, \\ 1 & \text{for } k \ge K_i \end{cases} \tag{4}$$

where u_k^i is the action at the center of μ_k^i.

B. Optimistic optimization

SOOP is based on simultaneous optimistic optimization (SOO), while the closest related planning method, LP, is based on deterministic optimistic optimization (DOO). Next, DOO and SOO are introduced [11].

The problem is to maximize some function $f : X \to \mathbb{R}$, assumed locally Lipschitz around an optimum x^*:[3]

$$f(x^*) - f(x) \le l(x, x^*) \quad \forall x \in X \tag{5}$$

where $x^* \in \arg\max_{x \in X} f(x)$ and $l : X \times X \to [0, \infty)$ is a semimetric (which for convenience incorporates the Lipschitz constant). The optimization proceeds by hierarchically partitioning the domain X. This partitioning is represented by a tree structure $\mathcal{T}$ in which each node (d, i) is labeled by a point $x_{d,i}$ and represents a subset of X denoted $X_{d,i} \ni x_{d,i}$. Here,

[2]Terminal states can be used to represent e.g. "goal achieved" and "failure" situations. Whatever the action applied in a terminal state, the system cannot escape it and the reward is always zero.

[3]For the duration of Section II-B only, we reuse notations f, X, and x to mean the optimized function, function domain, and a point in the domain.

$d \geq 0$ is the depth in the three and i is the node index at a given depth. The root of the tree represents the entire domain X, and the tree is defined so that the children of a node form a partition of the set represented by their parent. The partitioning procedure must ensure, roughly speaking, that the diameters of all sets at a certain depth are equal, and that the diameter sequence $\Delta(d) = \sup_{x \in X_{d,i}} l(x_{d,i}, x), \forall i$ is decreasing with the depth, e.g. exponentially. (More precisely, the diameters at any d are only required to be upper-bounded by $\Delta(d)$, and of the order $\Delta(d)$, but since the idea is the same we explain things more simply by taking the diameters at d equal.) Furthermore, from a computational perspective the partitioning should be easy to generate and set diameters easy to compute. Fig. 2 exemplifies such a partitioning. Finally, the set of leaves of the currently explored tree is denoted by $\mathcal{L}$.

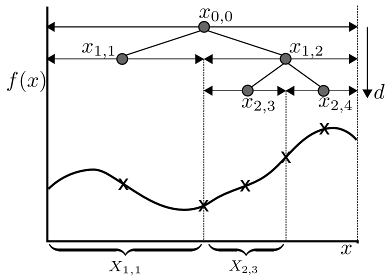

Fig. 2. Illustration of the tree structure that is used by optimistic optimization. In this example, X is an interval and binary partitions are used.

DOO works by partitioning a set that may contain the optimum of f. It does this by assigning upper bounds to all leaf sets $X_{d,i}, (d,i) \in \mathcal{L}$:

$$b(X_{d,i}) = f(x_{d,i}) + \Delta(d) \tag{6}$$

so that $b(X_{d,i}) \geq f(x),\ \forall x \in X_{d,i}$. Then at each iteration, an *optimistic* leaf set, which maximizes the upper bound, is further partitioned. At the end, the point with the largest value in the tree is returned. DOO is summarized in Algorithm 1.

Algorithm 1 Deterministic Optimistic Optimization

```
Input: function $f$, computation budget $n$, partitioning of $X$
1: initialize $\mathcal{T}$ with root $X_{0,0}$
2: for $t = 1$ to $n$ do
3:     $(d^*, i^*) \leftarrow \arg\max_{(d,i) \in \mathcal{L}} b(X_{d,i})$
4:     expand $(d^*, i^*)$ (partition $X_{d^*,i^*}$), add children to $\mathcal{T}$
5: end for
Output: $x^* = \arg\max_{(d,i) \in \mathcal{T}} f(x_{d,i})$
```

DOO assumes knowledge of l by using $\Delta(d)$ in the upper bounds. The alternative, SOO, does not require this knowledge. Instead, at each round, SOO simultaneously expands all potentially optimal leaf sets: those for which the upper bound could be largest under *any* semimetric l satisfying the conditions. With a little thought, a set can only contain a largest upper bound if its sample value is at least as good as the values of all sets with diameters larger than its own; we say that the set is not dominated by larger sets. Since further, $\Delta(d)$ decreases with d, we only have to compare with leaves higher up the tree. At each iteration t, the algorithm expands at most one leaf set at each depth. If we define $\mathcal{L}_{\leq d}$ as the set of leaf nodes having depths $d' \leq d$, then a leaf (d,i) is only expanded if $f(x_{d,i}) = \max_{(d',i') \in \mathcal{L}_{\leq d}} f(x_{d',i'})$; if there are several such leaves one is chosen arbitrarily. This selection procedure is illustrated in Figure 3. SOO additionally limits the tree depth at each iteration t with a function $d_{\max}(t)$, a parameter of the algorithm that controls the tradeoff between deeper or more uniform exploration. Algorithm 2 summarizes SOO. Note that in the form given here, SOO may take more than the budget n to finish the last iteration.

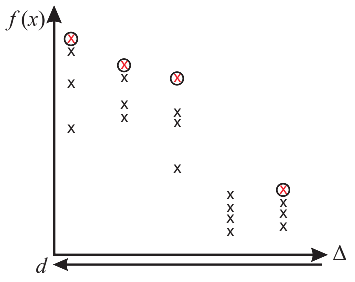

Fig. 3. Illustration of set selection in SOO. Depth d increases to the left, while set diameter Δ increases to the right. Samples are shown as 'x', and the samples of sets selected for expansion are circled and colored red.

Algorithm 2 Simultaneous Optimistic Optimization

```
Input: function $f$, depth function $d_{\max}(\cdot)$, budget $n$,
       partitioning of $X$
1: initialize $\mathcal{T}$ with root $X_{0,0}$; $t \leftarrow 1$
2: while $t \leq n$ do
3:     $f_{\max} \leftarrow -\infty$
4:     for $d = 0$ to $\min(\mathrm{depth}(\mathcal{T}), d_{\max}(t))$ do
5:         $i^* \leftarrow \arg\max_i f(x_{d,i})$
6:         if $f(x_{d,i^*}) \geq f_{\max}$ then
7:             expand $(d, i^*)$, add children to $\mathcal{T}$
8:             $f_{\max} \leftarrow f(x_{d,i^*})$
9:             $t \leftarrow t + 1$
10:        end if
11:    end for
12: end while
Output: $x^* = \arg\max_{(d,i) \in \mathcal{T}} f(x_{d,i})$
```

DOO and SOO have similar guarantees, converging to near-optimal solutions at rates depending on how "peaky" the function is around the optimum in the semimetric l. Of course, SOO pays a price for not knowing the semimetric by expanding several sets at each iteration. But not requiring the semimetric also has a surprising advantage: SOO converges at the fastest rate allowed by any semimetric. The difficult issue is, as the planning setting below will illustrate, to define the hierarchical partitioning in such a way that the diameters nicely decrease in the unknown semimetric.

C. Lipschitzian planning

LP [9] applies DOO to optimize the return (1) over the space U^∞ of infinite action sequences. The form of LP we introduce makes some mild changes to the version in [9], which we point out later. The dynamics f and rewards r are assumed to be Lipschitz with a known constant L:

$$\begin{aligned} \|f(x,u) - f(x',u')\| &\le L(\|x - x'\| + |u - u'|) \\ |r(x,u) - r(x',u')| &\le L(\|x - x'\| + |u - u'|) \end{aligned} \tag{7}$$

To apply DOO, first a semimetric l is needed. After some simple calculations that exploit the Lipschitz property, the difference between the rewards obtained at step k by two sequences $\mathbf{u}_\infty, \mathbf{u}'_\infty$ is bounded as follows:

$$|r(x_k,u_k) - r(x'_k,u'_k)| \le \sum_{j=0}^{k} L^{k-j+1}|u_j - u'_j|$$

Using this, we construct the semimetric as the following upper bound on the difference between the *returns* of the sequences:

$$l(\mathbf{u}_\infty, \mathbf{u}'_\infty) = \sum_{k=0}^{\infty} \gamma^k \min\{1, \sum_{j=0}^{k} L^{k-j+1}|u_j - u'_j|\} \tag{8}$$

where the reward difference bounds are additionally capped by 1 (recall that the rewards are bounded in $[0,1]$).

The partitioning scheme is trisection-based, as explained in Section II-A. Since the samples are infinite action sequences, the algorithm never has access to a complete sample or its value (the infinite-horizon return). Fortunately this is not a problem, because the metric l can still be used to provide an upper bound on the returns of sequences in a box discretized only up to some finite dimension $K_i - 1$:

$$b(\mathcal{U}_i) = \sum_{k=0}^{K_i-1} \gamma^k \min\{1, r^i_k + \sum_{j=0}^{k} L^{k-j+1}\delta^i_j\} + \frac{\gamma^{K_i}}{1-\gamma} \tag{9}$$

where r^i_k is the reward obtained at step k by applying the (finite) sequence $\mathbf{u}^i_{K_i}$ at the center of the box, and δ^i_k are the subspace sizes. Each term of the outer sum bounds the reward attainable at step k by any sequence in the box, while the fraction covers the reward attainable from step K_i onwards. The difference $b(\mathcal{U}_i) - R(\mathbf{u}^i_{K_i})$, see (3), can be informally thought of as the diameter of $\mathcal{U}_i$.

LP works by following the principle of DOO: at each iteration, it selects an *optimistic* box $\mathcal{U}_{i^*}$ that has the largest upper bound $b(\mathcal{U}_i)$, and further refines this box by trisecting one of its dimensions. To complete the algorithm, we only have to specify the dimension selection procedure. Each dimension $k < K_{i^*}$ in turn is assumed trisected, and the upper bound for the resulting middle box is computed, which will be smaller due to the reduced subspace size $\delta^{i^*}_k/3$. To rank the first undiscretized dimension K_{i^*}, the center reward is assumed to be 0, and the subspace size will be $1/3$. Finally, the selected dimension is one that reduces the bound the most.

Once an imposed budget n of calls to the model (f, r) has been depleted, the algorithm returns a center sequence with the largest return among all the boxes.

The original LP in [9] is different in the following ways. (i) The semimetric (8) and upper bound (9) are changed to cap individual reward bounds to 1 only after reaching the last k for which the reward bound is smaller than 1 (denote it by K'); thus (8) and (9) are tighter. (ii) If $K' < K_{i^*} - 1$ for the optimistic box, only dimensions up to K' are considered for trisection, whereas we still consider all dimensions including K_{i^*}. This avoids some pathological behavior such as when the first-step rewards r_0 are always 1, in which case the original LP would keep refining the first action dimension without ever going deeper. Finally, (iii) when a dimension $k < K_{i^*} - 1$ is trisected, we compute all the rewards up to $K_{i^*} - 1$ for the left and right center action sequences, whereas original LP only computes the kth rewards. This allows a fair comparison with SOOP, which trisects in the same way. It may either increase or decrease performance with respect to the original LP (increase because the initial upper bounds of the left and right boxes are tighter, decrease because more model calls are spent).

III. SOOP

Determining the Lipschitz L constant is hard, and, in fact, it must usually be treated as a tuning parameter of LP. Even so, f or r may simply not be Lipschitz. If they are, a global Lipschitz constant may underestimate their smoothness in large parts of the domain, leading to inefficient partitioning. Conversely, overestimating the smoothness (taking L too small) is dangerous because the upper bounds become invalid and the DOO guarantees are lost.

Therefore, we now propose an optimistic planning method that does not require a Lipschitz constant or knowing the semimetric, by exploiting the principles of SOO. Since the trisection scheme of LP is also used, many of the building blocks for the new method are already available. We still have to introduce the crucial insight that connects the pieces together into the overall, novel algorithm. We call this algorithm SOOP (Simultaneous Optimistic Optimization for Planning).

A. Potentially optimal boxes. Generic SOOP algorithm

The main step in SOO is selecting potentially optimal sets. This is ideally done by sorting the sets by their diameters, and then only selecting sets with values undominated by the values of larger-diameter sets. Note that the diameters themselves need not be known, only their ordering; in Algorithm 2, because diameter decreases with increasing depth, the depth d acts as a proxy for the ordering. Unfortunately, such a global ordering is very difficult to define for the planning problem. To address this difficulty, we relax the SOO set selection procedure.

First, because depth no longer translates into a diameter ranking, we stop looking at the sets as being organized into a tree. Instead, the algorithm just works with a collection of sets (boxes in the planning context), which does not affect its validity. We define a notion of partial ordering on these boxes, and impose an assumption. For any box $\mathcal{U}_i$, denote by $s^k_i \ge 0$ the number of times the box has been trisected along dimension k.

Definition 2: A box $\mathcal{U}_j$ is said to be partially greater than $\mathcal{U}_i$, denoted $\mathcal{U}_j \succeq \mathcal{U}_i$, iff $\forall k \geq 0$, $s_k^j \leq s_k^i$.

Assumption 3: If $\mathcal{U}_j \succeq \mathcal{U}_i$, then diameters $\Delta(\mathcal{U}_j) \geq \Delta(\mathcal{U}_i)$, where $\Delta(\mathcal{U}) = \sup_{\mathbf{u}_\infty, \mathbf{u}'_\infty \in \mathcal{U}} l(\mathbf{u}_\infty, \mathbf{u}'_\infty)$ is the box diameter in the unknown semimetric l.

We expect that many useful semimetrics will satisfy Assumption 3. For instance, it can be shown that the Lipschitz semimetric (8) satisfies it. Under Assumption 3, we modify the box selection procedure as follows: a box $\mathcal{U}_i$ is potentially optimal and will be expanded if it is undominated by any $\mathcal{U}_j \succeq \mathcal{U}_i$; that is, if for all $\mathcal{U}_j \succeq \mathcal{U}_i$, $R(\mathbf{u}_{K_i}^i) \geq R(\mathbf{u}_{K_j}^j)$. So, $\mathcal{U}_i$ will be compared only with *some* of the boxes with larger diameters: those that are partially greater than it. It will still be expanded if it is dominated by some larger box that is not partially greater. Thus the new criterion is safe (all boxes that should be expanded are indeed expanded) but conservative (some boxes that ideally should not be expanded perhaps will be). Conservativeness implies the algorithm requires more samples than an ideal application of SOO.

The final step is specifying how to select the dimension (action step) for trisection. Ideally, the dimension that leads to the largest decrease of the diameter in l should be trisected, but of course finding this decrease is not possible since l is unknown. We leave this procedure open in the general method, summarized as Algorithm 3, and discuss alternatives below. Note that the algorithm may take more than n transitions to complete the last iteration (expand the last batch of potentially optimal boxes).

Algorithm 3 SOO for Planning

```
Input: state $x_0$, model $(f, r)$, budget of model calls $n$
 1: initialize collection of boxes with $\mathcal{U}_1 = U^\infty$
 2: repeat
 3:     select potentially optimal boxes:
          $I^* = \{i \,|\, \forall\, j$ s.t. $\mathcal{U}_j \succeq \mathcal{U}_i, R(\mathbf{u}_{K_i}^i) \geq R(\mathbf{u}_{K_j}^j)\}$
 4:     for $i \in I^*$ do
 5:         select dimensions $\kappa \subseteq \{0, K_i\}$ to trisect
 6:         for $k \in \kappa$ do
 7:             trisect dimension $k$,
                  add resulting boxes to the collection
 8:         end for
 9:         remove parent $\mathcal{U}_i$ from the collection
10:     end for
11: until budget $n$ has been depleted
Output: best sequence found $\mathbf{u}_{K_{i^*}}^{i^*}$, $i^* \in \arg\max_i R(\mathbf{u}_i^{K_i})$
```

B. Dimension selection

Several alternatives for dimension selection are possible. (i) The simplest is to just trisect all dimensions $\{0, K_i\}$. This is safe, but costly in terms of model calls and computation.

Otherwise, one can conjecture that due to the discounting, which makes earlier actions more important, these actions should be discretized more finely. Thus a second alternative is to (ii) trisect those dimensions for which the resulting boxes are discretized more finely for smaller k, formally: $s_k^i \geq s_{k+1}^i \; \forall k \geq 0$. Then by induction, all boxes created by the algorithm satisfy the property. (iii) With the same conjecture, an even less costly heuristic may be derived that only selects one dimension. This is done by ranking dimensions with a new discount factor $\alpha \in (0, 1)$:

$$\kappa = \min\{\arg\max_{k \in \{0, K_i\}} \alpha^k (1/3)^{s_i^k}\} \tag{10}$$

The tuning parameter α trades off discretization accuracy and planning depth: small values will lead to finer discretizations close to the root, while with a larger value larger planning horizons are reached. In this sense, α is similar to the depth function $d_{\max}$ in SOO. With this criterion as well, all boxes produced are discretized more finely for smaller k.

Since in preliminary experiments trisecting many dimensions greatly increased computational costs without large performance benefits, we use (iii) in the sequel.

To extend the algorithm to multiple action variables, the partial ordering and the dimension selection must be changed. Denoting the action variable index by m, the partial ordering can be changed by requiring that *all* variables m at every step k are split at most as many times in $\mathcal{U}_j$ as in $\mathcal{U}_i$. Dimension selection can be performed by extending (10) to compare also between the variables at each k; thus a pair (k, m) that maximizes the discounted size would be selected, breaking ties in favor of small k and arbitrarily among m.

We close this section by discussing the amount of model calls required for trisections. Trisecting a box $\mathcal{U}$ of depth K along dimension k requires 3 model calls when $k = K$, and $2(K - k)$ if $k < K$. This is because in the former case all three new boxes inherit the entire center sequence $\mathbf{u}_K$ of $\mathcal{U}$, with the associated rewards, and must only simulate the next action (step K). When $k < K$, the center box retains again the complete information, whereas the left and right boxes only inherit the subsequence and rewards up to $k - 1$, and the tails from k to $K - 1$ must be simulated.

IV. Experiments

To determine the practical effectiveness of SOOP, it will be tested on three problems, in which it will be compared with three other state-of-the-art OP algorithms.

The first algorithm is OP for deterministic systems (OPD) [4], which serves as a discrete-action baseline. OPD applies DOO to search the space of infinite sequences of M *discrete* actions. The metric is $l(\mathbf{u}_\infty, \mathbf{u}'_\infty) = \gamma^{k(\mathbf{u}_\infty, \mathbf{u}'_\infty)}/(1 - \gamma)$, where $k(\mathbf{u}_\infty, \mathbf{u}'_\infty)$ is the first step where the two sequences are different. The hierarchical partition splits at each depth d the considered box along dimension $k = d$, into M subboxes: one for each discrete action. So, for OPD the depth in the tree is equal to the time step.

The other two algorithms support continuous actions: they are LP, the closest relative of SOOP, and HOLOP [10]. The latter is selected as a representative for the class of finite-horizon planning algorithms, which also includes HOOT [8]

TABLE I

ALGORITHM PARAMETERS FOR THE DC MOTOR, WITH ALL ATTEMPTED VALUES AS WELL AS THE BEST VALUES FOR EACH n.

	$n = 100$	$n = 500$	$n = 1000$	$n = 2500$	$n = 5000$
SOOP, $\alpha \in \{0.1, 0.2, \ldots, 0.9\}$	0.8	0.7	0.8	0.7	0.7
OPD, $M \in \{3, 5, \ldots, 15\}$	3	3	3	3	5
LP, $L \in \{0.1, 0.2, \ldots, 1.5\}$	0.9	0.6	0.6	0.7	0.5
HOLOP, $K \in \{5, 10, 15, 20, 25, 30, 40, 50, 75, 100\}$	5	5	5	5	5

and SP [9]. Unlike HOOT and SP, HOLOP solves a well-defined optimization problem over K-step action sequences.

For each problem, the algorithms are executed for several values of the budget n of model calls. Like for SOOP above, the algorithms are not stopped mid-iteration, so they may take more than n calls to complete. For each value of n, the other algorithm parameters are optimized over a grid, and the best performance is reported. The parameters are: for SOOP, the discount factor α for dimension selection; for OPD, the number of discrete actions M (for every M, a uniform grid of actions is generated, covering the whole action space); for LP, the Lipschitz constant L; and for HOLOP, the horizon K. Since HOLOP generates solutions randomly, it is run 10 times for each experiment and a 95% confidence interval on the mean performance is computed. The best experiment is the one with the largest upper confidence bound.

A. DC motor

The first problem concerns a DC motor with states: shaft angle $x_1 \in [-\pi, \pi]$ rad, angular velocity $x_2 \in [-15\pi, 15\pi]$ rad/s, and action: voltage $u \in [-10, 10]$ V. The dynamics are linear:

$$f(x, u) = Ax + Bu, \quad A \approx \begin{bmatrix} 1 & 0.0095 \\ 0 & 0.9100 \end{bmatrix}, \; B \approx \begin{bmatrix} 0.0084 \\ 1.6618 \end{bmatrix}$$

The goal is stabilizing both states at zero, and is described by the unnormalized reward function:

$$\tilde{r}(x, u, x') = -x^T Q x - u^T R u, \; Q = \mathrm{diag}(1, 0.001), \; R = 0.05 \tag{11}$$

with discount factor $\gamma = 0.95$. Using the known variable bounds, the reward is normalized (scaled and translated) into $[0, 1]$, and for the sake of applying the continuous-action algorithms, the same is done for the action.

This first problem is chosen because it is simple and can be solved with short planning horizons. Nevertheless, continuous (or finely discretized) actions are necessary for good performance, due to the quadratic action penalty. The four planning algorithms are applied in receding horizon, from the initial state $[-\pi, 0]^T$ and for a duration of 1 s (100 steps). Table I shows the parameters of the algorithms, where each row corresponds to an algorithm, and each column to a budget value n. The header column shows all values attempted for the algorithm's parameter (these are the same for all other problems, as well, so they will not be shown again), while the other columns show the best value for the corresponding n. Figure IV-A shows the best returns obtained.

SOOP is clearly better than OPD, as expected from the fact that coarse actions are not sufficient. An interesting observation is that despite this, discretizing finely is not worth

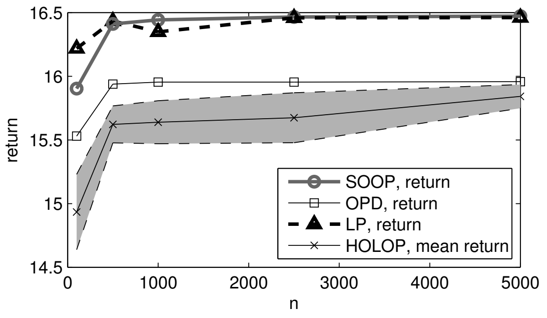

Fig. 4. Performance for the DC motor. For HOLOP, the mean performance with its 95% confidence interval is shown.

the additional price paid in terms of model calls in OPD (since a larger tree must be explored), not even for larger budgets. Only for $n = 5000$ do we get better performance by taking $M = 5$ discrete actions.

SOOP and LP are performing similarly: LP is better for small budgets, while SOOP overtakes it for larger ones. Apparently, a global Lipschitz assumption works in this problem, which is not surprising due to its simplicity.

HOLOP is doing worse than all others, and looking at controlled trajectories (not shown here due to space limitations) this is due to very coarse actions which are not able to stabilize the system. Thus, for the budgets considered here, HOLOP cannot sufficiently refine the solution.

B. Inverted pendulum swingup

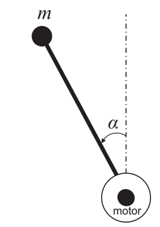

Fig. 5. Inverted pendulum schematic.

The second problem is swinging up and stabilizing an underactuated inverted pendulum rotating in a vertical plane, see Figure 5. Due to limited power, from certain states (e.g., pointing down) the pendulum needs to be swung back and forth to gather energy, prior to being pushed up and stabilized. The first state $x_1 = \alpha$ is the angle and wraps around in the interval $[-\pi, \pi)$ rad; the second state is the angular velocity $x_2 = \dot{\alpha} \in [-15\pi, 15\pi]$ rad/s. The action $u \in [-3, 3]$ V is the motor voltage (see [2], Section 4.5.3 for the dynamics).

The goal of stabilizing the pendulum pointing up is expressed by quadratic rewards of the form (11) with $Q = \text{diag}(1, 0)$, $R = 0.3$, and the discount factor is $\gamma = 0.95$. Like before, rewards and actions are normalized into $[0, 1]$.

While it is a standard benchmark in control and dynamic programming, this problem nevertheless supplies an interesting challenge to planning algorithms: the solution must be planned over a longer horizon, and solutions that seem good over a short horizon will not work, instead just pushing the pendulum in one direction. Furthermore, continuous actions are necessary, firstly due to the action penalty, and secondly to properly stabilize the pendulum in the unstable, pointing-up position. The planning algorithms are applied from an initially pointing down position, $x = [-\pi, 0]^T$, for a duration of 5 s (100 steps). Table II shows the algorithm parameters and Figure IV-B the best returns.

TABLE II
PARAMETERS FOR THE INVERTED PENDULUM.

$n =$	500	1000	5000	10000	15000
SOOP, $\alpha =$	0.9	0.8	0.7	0.7	0.7
OPD, $M =$	3	3	3	3	5
LP, $L =$	0.1	0.2	0.1	0.1	0.7
HOLOP, $K =$	10	10	10	10	10

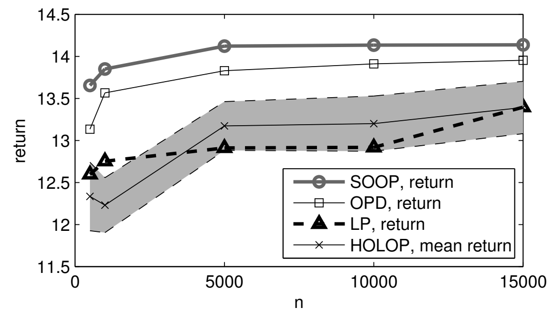

Fig. 6. Performance for the inverted pendulum.

The relationships between SOOP, OPD, and HOLOP mirror those in the DC motor problem. However, LP now ranks as poorly as HOLOP. Figure 7 (on the next page) shows representative controlled trajectories with SOOP and LP. LP applies very coarse actions, while SOOP uses fine discretization to behave near-optimally.[4] The reason is found in the small values of L: LP prefers to search longer-horizon solutions rather than discretize finely. Unfortunately, even for this coarse discretization it does not manage a good swing-up. While the reasons are not entirely clear, one hypothesis is that unlike for the DC motor, in the swing-up problem the Lipschitz constant varies, with the system behaving differently around equilibria than around the critical swing-up points; and that LP cannot deal with that.

Regarding α in SOOP, for tight budgets larger values are preferred, which means a longer horizon is sought at the expense of discretization; as more samples become available and a sufficient horizon is ensured, the balance shifts back towards discretization. This behavior is intuitive, since for too short horizons a good swing-up cannot be achieved, and fine actions become irrelevant.

[4]This is determined by comparing with near-optimal solutions found with dynamic programming, which is possible in this low-dimensional problem.

Finally, we look at the computational cost of the algorithms, see Figure IV-B. Besides the fact that in our Matlab implementation the algorithms are not yet ready for real-time control, we notice that SOOP and OPD have similar costs, and HOLOP is somewhat faster. LP is slower, but this is at least partly due to our implementation, which is optimized for many sequences with similar lengths; whereas around the goal state, LP typically expands a few very long sequences.

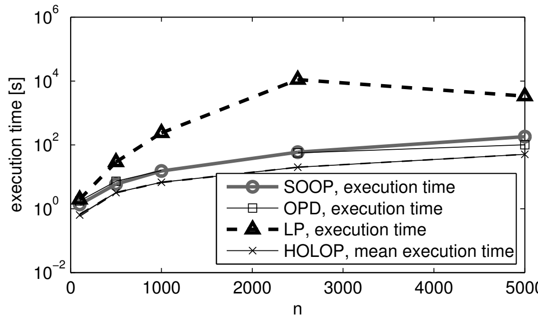

Fig. 8. Execution time for the inverted pendulum, for optimized parameters.

C. Two-link robot arm

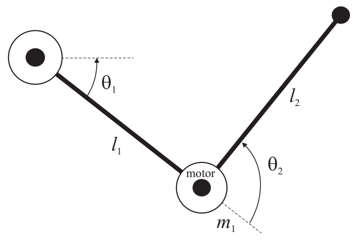

Fig. 9. Robot arm.

Finally, we consider a two-link robot arm actuated only in the middle joint, which has 4 states (angles θ_1, θ_2 of the joints plus their angular velocities) and 1 action u (motor torque). It can also be seen as a horizontally-oriented acrobot. The model equations are found in [2], Section 4.5.2. The link lengths are 0.15 and 0.25 m, both masses are 1 kg and concentrated at the ends of the links, and there is neither inertia nor friction. The task is stabilization to zero starting with both links at rest at angle $-\pi$, and the reward is quadratic with $Q = \text{diag}(1, 0, 1, 0)$ and no action penalty. Table III and Figure IV-C show the results. that OPD and discrete actions do well also in this problem, with SOOP trailing closely behind and doing better than LP and HOLOP.

Looking at Table III and Figure IV-C, OPD and discrete actions do well in this problem, with SOOP trailing close behind. Note that in problems where discrete actions work well, SOOP cannot be expected to outperform OPD, mainly because OPD searches the smaller space of discrete-action sequences,

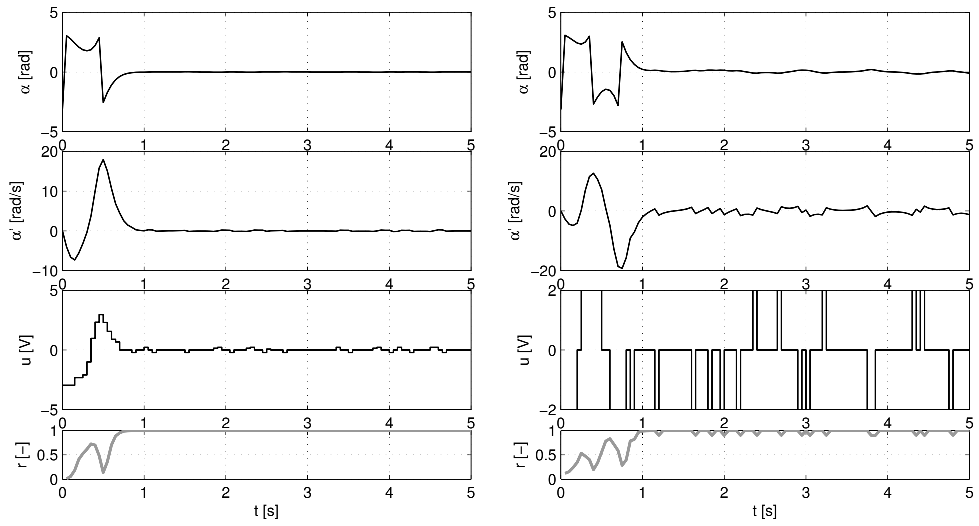
Fig. 7. Swing-ups of the inverted pendulum with SOOP and LP, for $n = 5000$ and optimized parameters.

which still contains a good solution. Nevertheless, here SOOP still manages to find a good solution in the larger, continuous-action space, obtaining similar performance to OPD and still outperforming LP and HOLOP, which apparently search the larger space less efficiently.

TABLE III
PARAMETERS FOR THE TWO-LINK ROBOT ARM.

$n =$	500	1000	5000
SOOP, $\alpha =$	0.5	0.6	0.5
OPD, $M =$	5	7	7
LP, $L =$	0.1	0.9	1.1
HOLOP, $K =$	5	5	5

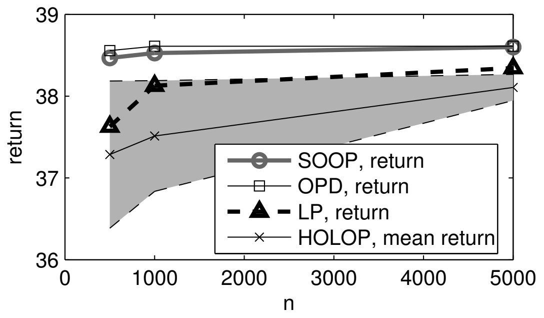

Fig. 10. Performance for the two-link robot arm.

V. Conclusions

We introduced *SOO for Planning*, a novel planning algorithm for deterministic, continuous-action Markov decision processes. In extensive experiments, SOOP consistently ranked among the best algorithms, fully dominating competing methods when the problem requires both long horizons and fine discretization. In problems where discrete actions do well, discrete-action planning starts at an advantage; nevertheless, in our example that had this property, SOOP could still be applied with minimal loss of performance, unlike its continuous-actions competitors.

The main open issue to address in future work is analyzing the performance of SOOP as a function of the budget n of model calls, which should build on the analysis of SOO [11]. In addition, more experiments are required to better understand the effect of the various dimension selection criteria.

References

[1] D. P. Bertsekas, *Dynamic Programming and Optimal Control*, 3rd ed. Athena Scientific, 2007, vol. 2.

[2] L. Buşoniu, R. Babuška, B. De Schutter, and D. Ernst, *Reinforcement Learning and Dynamic Programming Using Function Approximators*, ser. Automation and Control Engineering. Taylor & Francis CRC Press, 2010.

[3] 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.

[4] 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.

[5] S. Bubeck and R. Munos, “Open loop optimistic planning,” in *Proceedings 23rd Annual Conference on Learning Theory (COLT-10)*, Haifa, Israel, 27–29 June 2010, pp. 477–489.

[6] T. J. Walsh, S. Goschin, and M. L. Littman, “Integrating sample-based planning and model-based reinforcement learning,” in *Proceedings 24th AAAI Conference on Artificial Intelligence (AAAI-10)*, Atlanta, US, 11–15 July 2010.

[7] L. Buşoniu and R. Munos, “Optimistic planning for Markov decision processes,” in *Proceedings 15th International Conference on Artificial Intelligence and Statistics (AISTATS-12)*, ser. JMLR Workshop and Conference Proceedings, vol. 22, La Palma, Canary Islands, Spain, 21–23 April 2012, pp. 182–189.

[8] 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.

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

[10] A. Weinstein and M. L. Littman, “Bandit-based planning and learning in continuous-action Markov decision processes,” in *Proceedings 22nd International Conference on Automated Planning and Scheduling (ICAPS-12)*, São Paulo, Brazil, 25–19 June 2012.

[11] 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.