Optimistic Planning with Long Sequences of Identical Actions for Near-Optimal Nonlinear Control

Koppány Máthé, Lucian Buşoniu, Liviu Miclea
Department of Automation, Technical University of Cluj-Napoca, Memorandumului 28, 400114 Cluj-Napoca, Romania
{koppany.mathe, lucian.busoniu, liviu.miclea}@aut.utcluj.ro

***Abstract*—Optimistic planning for deterministic systems (OPD) is an algorithm able to find near-optimal control for very general, nonlinear systems. OPD iteratively builds near-optimal sequences of actions by always refining the most promising sequence; this is done by adding all possible one-step actions. However, OPD has large computational costs, which might be undesirable in real life applications. This paper proposes an adaptation of OPD for a specific subclass of control problems where control actions do not change often (e.g. bang-bang, time-optimal control). The new algorithm is called Optimistic Planning with K identical actions (OKP), and it refines sequences by adding, in addition to one-step actions, also repetitions of each action up to K times. Our analysis proves that the a posteriori performance guarantees are similar to those of OPD, improving with the length of the explored sequences, though the asymptotic behaviour of OKP cannot be formally predicted a priori. Simulations illustrate that for properly chosen parameter K, in a control problem from the class considered, OKP outperforms OPD.**

I. Introduction

Optimal control methods focus on finding the best control for a given problem by minimizing certain costs (e.g. process time reduction, minimization of energy consumption, etc.). Optimistic planning for deterministic systems (OPD) [1] is one such algorithm, designed for deterministic control problems with control input sets consisting of finitely many discrete values. OPD works on the principle of constructing a search tree of actions that form action sequences starting from the root node. The algorithm iteratively builds these sequences by always adding single actions to the most promising sequence. At the end of the search, OPD chooses the first action from the best-performing action sequence and applies it to the current state of the system. Hence, it is a type of model-predictive control [2].

OPD is able to deal with highly nonlinear systems, ensuring computational and near-optimality guarantees. It is also optimal, roughly meaning that for any algorithm, a problem can be found where this algorithm will do at most as well as OPD. This extreme generality comes however at the cost of slow performance growth with the computation invested. Further, optimality in the sense above does *not* mean one cannot do better in specific classes of problems – an essential feature in practical applications. Such an improvement is our goal in the present paper. Specifically, we propose a specialization of optimistic planning targeting problems where longer ranges of constant control actions are preferred, so actions change only rarely. Numerous applications requiring time-optimal control (e.g. in vehicle path planning, aerospace) and bang-bang solutions fit this subclass [3]. Note that several other types of optimistic planning algorithms have been proposed, such as for stochastic problems [4], [5] and continuous action spaces [6], [7]. These all focus on generalizing OPD, while to our knowledge no work has been done to improve performance in specific types of problems.

The new algorithm we propose is called Optimistic Planning with K identical actions (OKP). Whereas in OPD the search refinement consists of adding all possible one-step actions to a chosen sequence, in OKP we are also adding the actions multiple times, up to K repetitions. This exploits the prior knowledge about the type of solution preferred. From the analysis, the near-optimality bound of OKP is similar to the one of OPD, resulting from the depth of the search tree. However, due to the variable depth in expansions introduced by K, one cannot usefully predict the performance of the algorithm a priori, as it could be done in case of OPD. In our simulations, this property of OKP leads to shallower search trees than those of OPD. Still, despite the shallower trees, in practice OKP outperforms OPD for the chosen control problem, and we expect that, for properly tuned parameter K, these results are valid for a general class of problems.

Related work includes piecewise-constant control [8] and specifically piecewise-constant model-predictive control [9], [10], as well as applications of related ideas to e.g. sampled-data systems [2]. Closer to our method, due to their focus on improving performance and reducing computation, are references [11]–[13]. In contrast to these works, where the analysis largely deals with stability, our theoretical discussion explicitly focuses on the relation between computation and performance of the algorithm. Also, unlike [13], our algorithm does not constrain where the action should be changed (although it works better when switches happen after around K steps).

The paper continues by introducing in Section II the concepts and brief analysis of optimistic optimization and planning, which form the basis of OKP. Section III presents the newly developed algorithm and its analysis, while Section IV shows the simulation results comparing OKP to OPD. Section V concludes the paper.

II. Theoretical Background

OKP is an extension of OPD, whereas OPD is based on the concepts of Optimistic Optimization for Deterministic Functions (OOD). Therefore, we first introduce the main concepts of the OOD method.

A. Optimistic Optimization for Deterministic Functions

OOD [1] is an optimization method that searches for the maximum of a function $v : \mathcal{H} \to \mathbb{R}$, whose form is not known

The work of K. Máthé and L. Buşoniu was supported by a grant of the Romanian National Authority for Scientific Research, CNCS-UEFISCDI, project number PNII-RU-TE-2012-3-0040.

but which can be sampled in some points $v(h)$. The search is performed by iteratively splitting the set $\mathcal{H}$ into subsets H_d. This splitting or partitioning procedure can be represented using a search tree having as nodes the sets H_d where d marks the depth of a node in the tree, as shown in Fig. 1. Note that H_d generally marks any node from depth d.

The assumptions we make regarding the form of the function and the search rules are:

- ***A1.*** *There exists a semi-metric l, i.e. a function $l : \mathcal{H} \times \mathcal{H} \rightarrow \mathbb{R}^+$, with the properties $l(h, \overline{h}) = 0$ iff $h = \overline{h}$ and $l(h, \overline{h}) = l(\overline{h}, h)$, for any $h, \overline{h} \in \mathcal{H}$.*
- ***A2.*** *There exists at least a global maximizer $h^* \in \mathcal{H}$ for which function v is locally one-sided Lipschitz around h^*, i.e. $v(h^*) - v(h) \leq l(h^*, h)$, for any $h \in \mathcal{H}$.*
- ***A3.*** *There exists a decreasing sequence $\delta(d) > 0$ such that for any depth d in the search tree and for any set H_d, diam$(H_d) \leq \delta(d)$, where diam$(H_d) = \sup\limits_{h,\overline{h} \in H_d} l(h, \overline{h})$ defines the diameter of a set $H_d \in \mathcal{H}$.*
- ***A4.*** *There exists a value $\nu > 0$ such that for any depth d, any set H_d contains an l-ball[1] of radius $\nu\delta(d)$ centered in the sampling point $h_s \in H_d$.*

In other words, assumptions A1 and A2 require a certain level of smoothness of function v, but otherwise v can be highly nonlinear. Assumptions A3 and A4 are related to the partitioning rule of the search algorithm, ensuring that the diameters of the search sets H_d decrease with the depth, but the algorithm will not end up partitioning a set into infinitely thin but large length subsets.

Based on assumption A2, for any set $H_d \subset \mathcal{H}$ containing the global maximizer h^*, given a sample h_s from the set, an upper bound on the values of points $h \in H_d$ can be defined:

$$b(H_d) = v(h_s) + \text{diam}(H_d) \tag{1}$$

with the property:

$$b(H_d) \geq v(h), \forall h_s, h \in H_d \tag{2}$$

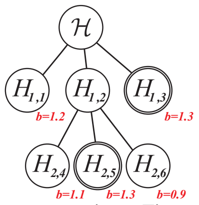

Fig. 1. Search tree after two expansions. First, root node $\mathcal{H}$ was expanded by splitting it into three subsets. Next, node $H_{1,2}$ was expanded. Now, upper bounds are calculated for the leaf nodes. Double circles mark the candidates for the next expansion. The method will choose arbitrarily one of these. Note that, in this example, each expansion means splitting a set into three subsets $H_{d,i}$. Index d marks the depth of the node in the tree while i tells the order of node insertions.

Having all these defined, the OOD method searches for the maximum of v by iteratively splitting the search set $\mathcal{H}$ into subsets H_d, taking a sample $v(h_s)$ for each subset, calculating the corresponding upper bounds and splitting further the set with the maximum upper bound. If there are several sets having the maximum upper bound, the method will choose arbitrarily one of them for further expansion.

[1]An l-ball is a set of points $B(h, \epsilon) = \{h' \mid l(h, h') \leq \epsilon\}$.

Note that the upper bound $b(H_d)$ makes sense for all the expanded nodes of the tree: having $b(H_d) \geq v^*$ for any node containing h^* implies that nodes having $b(H_d) < v^*$ are not expanded (since the expansion criterion will always choose from nodes with the highest upper bound and there is always at least a node containing h^*, which has $b(H_d) \geq v^*$). So, all the expanded nodes are likely to contain the optimum.

Using this approach, the optimization problem can be solved. The method is called optimistic due to the way it searches for the solution: it expands its search for new samples always in the direction of the most promising node (having the highest upper bound).

Note that since the search tree is theoretically infinite, the method has to limit the search. The computational budget n is a number telling the algorithm after how many node expansions it should stop the search. The quality of the result is improved with increasing n.

By limiting the search, the method may not reach the optimal point h^*. The near-optimality of the method is a measure of how close the result is to the optimal solution, expressed using the so called simple regret:

$$r_n = v(h^*) - v(h_s(n)) \tag{3}$$

where $h_s(n)$ marks the greedy sample, i.e. the point for which v has the highest value from the set of available samples after n expansions. Note that n need not be specified in advance, and by interrupting the algorithm at any time, the available solution makes sense: the simple regret is small enough, and it decreases with n.

The following theorem describes the performance of OOD.

Theorem 1. Under assumptions A1-A3, the near-optimality of the method can be bounded a posteriori as $r_n \leq \delta(d_{\max})$, where $d_{\max}$ marks the depth of the deepest expanded node. Under assumptions A1-A4, an a priori[2] near-optimality bound is $r_n \leq O(n^{-\frac{1}{\beta}})$, where β is the near-optimality dimension of the function v.

Proof (sketch). Regarding the a posteriori bound, using the expression of the simple regret from (3), assumption A3 and the fact that $v(h_{\max}) \leq v(h_s(n))$ (that is the value of the function v for the deepest sampling point $h_{\max}$ is less than or equal to the value for the greedy sample), one can write:

$$r_n = v(h^*) - v(h_s(n)) \leq v(h^*) - v(h_{\max}) \leq \delta(d_{\max}) \tag{4}$$

The a priori bound is discussed in more details in [1]. The intuition is that due to the exponential growth of the search tree, the near-optimality of the algorithm, derived from the depth reached in the search tree, will be improved polynomially with respect to the number of expansions n. The order of the polynomial is related to the so-called near-optimality dimension, $\beta > 0$, which characterizes the complexity of the problem: the smaller β, the simpler the problem and the faster the regret shrinks with n.

[2]The a priori bound means information available before applying the method, based on knowing the available computational budget n, while the a posteriori bound requires running the search.

B. Markov Decision Processes

OOD is a general method that may consider any type of function, search space, metric and partitioning rule under the described assumptions. OPD is an algorithm that applies the OOD principle to optimal control problems: for example, considering a car driving from a city to another, OPD would deal with finding the optimal direction to drive in at each intersection in order to minimize the travel time.

We model the control problem as a Markov Decision Process, which in the deterministic case consists of states x, actions u, a transition function $f(x, u)$ and an associated reward function $\rho(x, u)$. The space of possible states is denoted X, while the action space is denoted U. An action u makes it possible to move from a state to another. Function $f : X \times U \to X, f(x, u) = x'$ describes this transition from a state x to x' by applying an action u (i.e. the dynamics of the system). Each transition is evaluated by the reward function $\rho : X \times U \to \mathbb{R}, \rho(x, u) = r$, a measure of the quality of the transition.

Further notions used in the sequel are policy and return. A rule associating an action to each state, and thus describing the controller behaviour, is called a policy $\pi : X \to U$. The return is defined as:

$$V^{\pi}(x_0) = \sum_{k=0}^{\infty} \gamma^k \rho(x_k, \pi(x_k)) \tag{5}$$

which is the discounted sum of the rewards obtained while applying actions $\pi(x_k)$ as indicated by the policy, starting from state x_0 (note that $x_{k+1} = f(x_k, \pi(x_k))$). Note that $\gamma \in (0, 1)$ is the discount factor and optimistic planning is adapted to such discounted optimal control problems.

C. Optimistic Planning for Deterministic Systems

Optimistic Planning for Deterministic Systems (OPD) is a planning algorithm for Markov Decision Processes that applies the OOD method to find the optimal control action for a given state of a system.

State space X may have any structure. Regarding the action space U, it is assumed to be finite and discrete, $U = \{u^1, ..., u^M\}$. Also, the system dynamics f is assumed to be known, while the reward function $\rho(x_k, u_k)$ must be bounded and scaled (if needed) to the interval $[0, 1]$.

OPD optimizes the return over the space of infinite action sequences $h = [u_0, u_1, ...], h \in \mathcal{H}$. Thus, the function being optimized is:

$$v(h) = V^h(x_0) = \sum_{k=0}^{\infty} \gamma^k \rho(x_k, u_k) \tag{6}$$

Note that in this way the optimal action sequence h^* is sought rather than the optimal policy function π^*. Though this extra generality is not necessary, it is more convenient to associate such sequences to the search tree of the algorithm.

The search set $\mathcal{H}$ thus is the set of infinitely long action sequences:

$$\mathcal{H} = \{h = [u_0, u_1, ..., u_k, ...], u_k \in U\} \tag{7}$$

The root node of the search tree has this set assigned to it. A node at depth d of the tree is assigned a subset H_d containing action sequences known only up to depth d. Expanding a node means fixing another action in the sequences in H_d. Thus, a subset H_d has the form:

$$H_d = \{[u_0, u_1, ..., u_{d-1}, \bullet, \bullet, ...]\} \tag{8}$$

where u_{d-1} marks an action that leads to depth d of the tree and the sign $\bullet$ indicates any action can be taken from the space U.

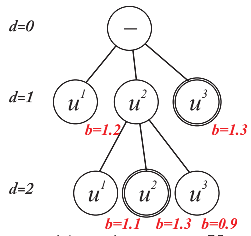

Fig. 2. OPD search tree with action space $U = \{u^1, u^2, u^3\}$. Actions leading to nodes are put in the nodes. The nodes of the tree are: $\mathcal{H}$, $\{[u^1, \bullet, \bullet, ...]\}$, $\{[u^2, \bullet, \bullet, ...]\}$, $\{[u^3, \bullet, \bullet, ...]\}$, $\{[u^2, u^1, \bullet, \bullet, ...]\}$, $\{[u^2, u^2, \bullet, \bullet, ...]\}$, $\{[u^2, u^3, \bullet, \bullet, ...]\}$.

The partitioning rule (node expansion) of OPD divides a set into M subsets, one set for each possible action. Node expansion is performed using upper bounds similar to those presented with OOD. Thus, define the metric in OPD as:

$$l(h, \overline{h}) = \frac{\gamma^{\delta(h, \overline{h})}}{1 - \gamma} \tag{9}$$

with γ the discount factor and $\delta(h, \overline{h})$ the smallest index for which the two action sequences differ. Note that for any pair $h, \overline{h}$ from a set H_d, $\min_{h, \overline{h} \in H_d} \delta(h, \overline{h}) = d$, from where, using the definition of the diameter, $\text{diam}(H_d) = \frac{\gamma^d}{1-\gamma}$. Also, this metric defines the largest possible difference in the return of two action sequences from a set H_d, as the sequences are identical up to index $d - 1$ from where on the maximum difference between the returns is at most 1 for each action taken.

OPD cannot use $v(h_s)$, as in reality any sequence $h_s \in H_d$ is infinitely long and known only up to length d. Instead it makes the use of a lower bound on the value function of sequences in H_d:

$$\ell(H_d) = \sum_{k=0}^{d-1} \gamma^k \rho(x_k, u_k) \tag{10}$$

using which, the upper bound of OPD is defined as:

$$b(H_d) = \ell(H_d) + \frac{\gamma^d}{1 - \gamma} \tag{11}$$

Note that this is a valid upper bound in OPD, since, for any $h \in H_d$, having all the rewards in the range $[0, 1]$:

$$v(h) = \ell(H_d) + \sum_{k=d}^{\infty} \gamma^k \rho(x_k, u_k) \leq \ell(H_d) + \sum_{k=d}^{\infty} \gamma^k = b(H_d) \tag{12}$$

Having all these defined, after constructing the search tree in the same way as in OOD, OPD will choose as the solution the first action from the set H_d for which $\ell(H_d)$ is the highest.

Since OPD applies the OOD method, the analysis of the latter can be inherited if the assumptions of OOD hold for OPD as well.

Checking OPD against assumption A1 of OOD, the metric defined in (9) is a valid semi-metric as $l(h, \overline{h}) \geq 0$, $l(h, \overline{h}) =$

$l(\overline{h}, h)$ for any $h, \overline{h} \in \mathcal{H}$ and $\frac{\gamma^{\delta(h,\overline{h})}}{1-\gamma} = 0$ iff $h = \overline{h}$ (i.e. only when $\delta(h, \overline{h}) \to \infty$).

Looking at assumption A2, function v defined in (6) is locally Lipschitz, as by definition, the metric used in OPD equals the largest possible difference between the returns of h and h^*. Thus, for any $h \in \mathcal{H}$, $v(h^*) - v(h) \leq l(h^*, h)$.

Assumption A3 holds as well, as having $\text{diam}(H_d) = \frac{\gamma^d}{1-\gamma}$ and $\gamma \in (0, 1)$, $\delta(d) = \frac{\gamma^d}{1-\gamma} > 0$ is a decreasing sequence and $\text{diam}(H_d) = \delta(d)$, for any depth d and for any set H_d. Regarding the condition from assumption A4, it can be written as $\text{diam}(H_d) \geq \nu\delta(d)$, and it holds with $\nu = 1$.

With all the assumptions of OOD satisfied for OPD as well, Theorem 1 is valid for OPD too. Hence, the near-optimality r_n of the OPD algorithm can be bounded a posteriori as

$$r_n \leq \delta(d_{\max}) = \frac{\gamma^{d_{\max}}}{1-\gamma} \tag{13}$$

with $d_{\max}$ the depth of the deepest expanded node.

Also, an a priori regret bound of the form in Theorem 1 can be defined as a function of the computational budget n.

III. Optimistic Planning with K identical actions

Focusing on the subclass of control problems with ranges of identical control actions, optimistic planning with K identical actions (OKP) considers the OPD algorithm with the following change: at each expansion, create $M \cdot K$ children, K nodes corresponding to each M distinct action, one for repetition of each action from 1 to K times. OKP will select then the next candidate node for expansion in the same way as OPD did. Note that choosing $K = 1$, the algorithm reduces to OPD.

The novelty of OKP rests in looking with each search step at the repeated sequence of each distinct action. If a control problem prefers ranges of identical actions, intuitively, this algorithm should find faster a near-optimal action sequence and explore deeper on the optimal path than OPD.

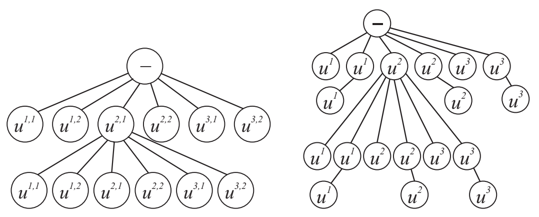

Fig. 3. OKP search tree with action space $U = \{u^1, u^2, u^3\}$ and maximum repetitions $K = 2$. On the left, the new tree is represented with actions of the form $u^{m,k}$ where m tells which action it is from the action space U, $m = 1, ..., M$, while index k tells its repetition, $k = 1, ..., K$. The figure on the right shows the same tree unwrapped, i.e. in the OPD representation.

The representation in the left graph will be used in the sequel: a node of the tree is e.g. $\tilde{H}_2 = \left\{\tilde{h} = [u^{2,1}, u^{3,2}, \bullet, \bullet, ...]\right\}$, with the action sequence in the unwrapped form $h = [u^2, u^3, u^3, ...]$.

The new structure introduces redundancy in the tree in the sense of duplicate action sequences: for instance, looking at Fig. 3, nodes $\left\{[u^{2,2}, \bullet, \bullet, ...]\right\}$ and $\left\{[u^{2,1}, u^{2,1}, \bullet, \bullet, ...]\right\}$ contain the same set of action sequences. Duplicates should not be created since these are unwanted. Details follow in the sequel. Next, similarly to the analysis from OPD, we check OKP against the assumptions of OOD.

Due to the different form of the action sequences, the metric of OKP is defined as:

$$\tilde{l}(\tilde{h}, \tilde{\overline{h}}) = \frac{\gamma^{\delta(\tilde{h}, \tilde{\overline{h}})}}{1-\gamma} \tag{14}$$

Note that $\tilde{l}(\tilde{h}, \tilde{\overline{h}}) = l(h, \overline{h})$ where $h, \overline{h}$ are the unwrapped representation of $\tilde{h}, \tilde{\overline{h}}$. In OKP $\delta(\tilde{h}, \tilde{\overline{h}}) = D = \sum_{i=1}^{d} k_i$ where D denotes the cumulative depth, with k_i the repetition of an action from depth i (e.g. if H_3 contains action sequences of the form $\tilde{h} = [u^{2,4}, u^{1,1}, u^{3,2}, ...]$, it will have $D = 4+1+2 = 7$). It is important to note that duplicate action sequences (i.e. different sequences $\tilde{h}, \tilde{\overline{h}}$ with the same unwrapped representation), would result in $\tilde{l}(\tilde{h}, \tilde{\overline{h}}) = 0$ and $\tilde{l}(\tilde{h}, \tilde{\overline{h}})$ would no longer be a semi-metric. Thus, it is required to eliminate duplicates in order to keep assumption A1 valid. The way this is performed is a detail of implementation that does not influence the analysis.

As OKP uses the same value function as OPD, the metrics from the two algorithms are equal and the unwrapped action sequences of OKP fully cover the set $\mathcal{H}$, there exists a global maximizer $\tilde{h}^* \in \tilde{\mathcal{H}}$ in OKP as well, for which for any $\tilde{h} \in \tilde{\mathcal{H}}$, $v(\tilde{h}^*) - v(\tilde{h}) \leq l(\tilde{h}^*, \tilde{h})$.

Also, as for OKP $\text{diam}(\tilde{H}_d) = \frac{\gamma^D}{1-\gamma}$ and the cumulative depth D increases with the depth in the wrapped tree, assumption A3 of OOD is valid as well. The decreasing sequence can be defined as $\delta(d) = \frac{\gamma^d}{1-\gamma}$, since the highest value of γ^D can appear for $D = d$, when actions are taken only once, as in OPD.

While it does not follow directly from Theorem 1 (which would only be able to ensure $\delta(d_{\max}) = \frac{\gamma^{d_{\max}}}{1-\gamma}$ near-optimality), under assumptions A1-A3 the following stronger result is shown in a very similar way:

Proposition 2. The near-optimality of the OKP algorithm can be bounded a posteriori as:

$$r_n \leq v^* - v(h_{D_{\max}}) \leq \frac{\gamma^{D_{\max}}}{1-\gamma} \tag{15}$$

where $D_{\max}$ is the largest cumulative depth of the expanded nodes at depth $d_{\max}$.

Regarding the asymptotic behaviour of OKP, since OKP adds with each depth nodes with 1 up to K actions, one cannot ensure that the diameter of the sets decreases in a good way. Stating this more formally, since $\delta(d) = \frac{\gamma^d}{1-\gamma}$, but on the other hand $\text{diam}(\tilde{H}_d) = \frac{\gamma^{K \cdot d}}{1-\gamma}$ in the worst case of a sequence having actions with repetition $k_i = K$ at each depth i, there exists no value $\nu > 0$ which would satisfy assumption A4 ($\text{diam}(\tilde{H}_d) \geq \nu\delta(d)$). Note that choosing $K = 1$ (original OPD) results in $\nu = 1$ which, as expected, validates assumption A4.

Without assumption A4, the number of possibly expanded nodes cannot be usefully bounded a priori and the near-optimality of the algorithm cannot be expressed as a tight function of the computational budget, like in the case of OOD and OPD. A worst-case a priori bound can be defined by considering uniform expansion of the tree, but this bound is not very useful as there is no dependence on the problem complexity (e.g. the near-optimality dimension β).

The a posteriori bound obtained for OKP might seem to be better than the one of OPD, as $D \geq d$ in the OKP trees. Still, this relation does not imply that for the same control problem OKP will reach deeper nodes in its search tree than OPD will. Thus, it might happen that $D_{OKP} < d_{OPD}$, i.e. that OKP will be able to ensure weaker performance than OPD will. Nevertheless, $D_{OKP} \geq d_{OPD}$ is expected for some classes of control problems where longer repeated actions are preferred.

Note that, like OPD, OKP returns an action sequence that could be applied in open loop. However, due to uncertainty, in practice it is better to only apply the first action of the sequence and then repeat the algorithm in receding horizon.

IV. Experimental results

The performance of the algorithms is compared using the problem of swinging up an inverted pendulum. The aim is to bring up and stabilize an underactuated pendulum to the pointing up state. Since, from certain states, the control power is not sufficient to bring up the pendulum using a single rotation, one or several swings may be required to bring the pendulum to the desired state. This requires a long planning horizon, ideal for the discussed algorithms. The bang-bang-like solutions (swings, that require switching between the two opposite maximum amplitude actions) should emphasize the benefits of OKP over OPD.

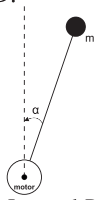

Fig. 4. Inverted Pendulum

The state vector consists of the angle of the pendulum and the angular velocity: $x = [x_1 \; x_2]^T$ with $x_1 = \alpha \in [-\pi, \pi)$ rad, $x_2 = \dot{\alpha} \in [-15\pi, 15\pi]$ rad/s. The angles wrap around in $[-\pi, \pi)$ and the angular velocity bounds are ensured by saturating it explicitly in the dynamics simulation. The action space consists of three discrete actions, $U = \{-2, 0, 2\}$ V that represent the motor voltage. The unnormalized reward function has a quadratic form: $\rho(x, u) = -x^T Q x - u^T R u$, with $Q = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}$ and $R = 0.1$. Using the known state bounds, the reward function is normalized to the interval $[0, 1]$. The discount factor is set to $\gamma = 0.98$.

Instead of directly using n, the computation budget telling the number of expansions, in our experiments we will use n_{sim}, representing the number of nodes added to the search tree (number of simulations of transition). In case of OPD $n_{sim} = M \cdot n$, whereas for OKP $n_{sim} = K \cdot M \cdot n$. Note that n_{sim} dominates computation rather than n.

We should further remark that since duplicates are eliminated from the OKP search tree, the algorithm may actually add fever nodes to the tree than $K \cdot M \cdot n$. To this end, an "OKP equal treesize" variant of the algorithm will be considered as well, that will add further nodes to the tree until reaching the same number of nodes n_{sim} as OPD.

The first set of experiments consists of offline simulations, that is, calculating a single control action for each starting state from a given set, here taken $\left\{-\pi, -\frac{5\pi}{6}, ..., \pi\right\}$ rad $\times$ $\{-15\pi, -14\pi, ..., 15\pi\}$ rad/s. With these tests the results of the algorithms can be directly compared by looking at the average of the real regret[3] obtained for these states.

After performing several tests, results indicate that OKP performs best for $K = 16$ for the problem considered. Using this value of the recurrence for OKP, Fig. 5 presents the regret for a set of n_{sim} between 600 and 7500. The main observation is that OKP in both its variants (the basic variant that will contain less nodes than $K \cdot M \cdot n$ due to duplicate elimination, and the "equal treesize" variant that considers expanding further nodes until reaching the same number of nodes as OPD) has better regrets than OPD. Besides this, there exists no clear difference between the results of the two variants. Note also that the performance of the algorithms may be nonmonotonic with the depth (and hence the budget), seen even with OPD.

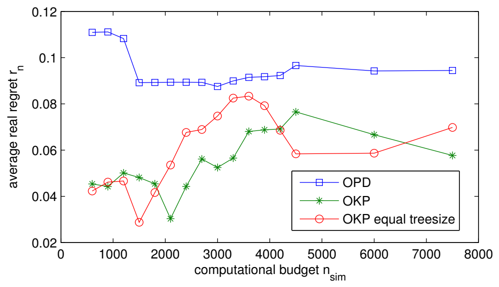

Fig. 5. Regret obtained with $K = 16$.

The next two figures show the average search tree depth. In the current context, the tree depth represents the depth of the deepest expanded node in case of OPD, while for OKP it refers to the cumulative depth D of the expanded node with the largest D. Note that this depth is used in the a posteriori regret bound, presented in (13) for OPD and (15) for OKP.

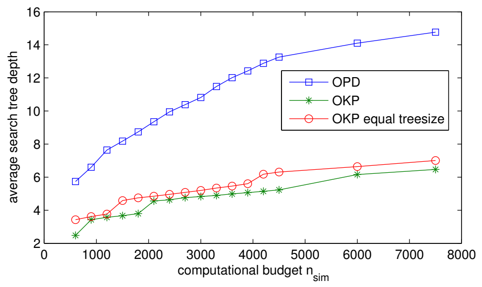

Fig. 6. Tree depth with $K = 16$.

Looking at Fig. 6, the increasing depths reflect the flow of the algorithms: more expansions lead to deeper trees. As OKP "equal treesize" expands more nodes than the other variant, this difference can be observed as larger depths for OKP "equal treesize". An important remark is that, unlike our

[3]The real regret is calculated using the optimum v^*, available from another algorithm at much higher computational costs

expectations, OPD manages to construct much deeper search trees than OKP is able to, i.e. $d_{OPD} > D_{OKP}$, as it was highlighted at the end of Section III. Our intuition was that OKP will find faster the optimal sequence and thus expand it deeper. However, this results invalidates the initial intuition, at least in our problem: from (13) and (15), the relation between the regret bounds is $r_{OKP} = \frac{\gamma^D}{1-\gamma} > \frac{\gamma^d}{1-\gamma} = r_{OPD}$ for $d > D$ (i.e. OPD has a better bound). However, in reality, as Fig. 5 shows, OKP obtains better real results than OPD. Generalizing this, for suitable control problems the practical performance of OKP may be better.

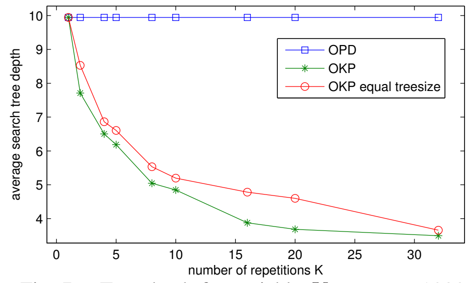

Fig. 7. Tree depth for variable K, $n_{sim} = 1920$.

Fig. 7 tells that increasing the maximum number of recurrence K of an action results in search trees with smaller depth, i.e. weaker performance guarantees. Thus, one should find a balance in the choice of K. Note that choosing $K = 1$ all the algorithms reach the same depth, as expected.

Another set of experiments tests the algorithms online. Given the starting state $x_0 = [-\pi\ 0]^T$ (i.e. pendulum pointing down, with zero angular velocity), a control action is calculated for it and applied to the pendulum, reaching a new state. In the new state, the algorithm is repeated and so on. The simulation lasts $T = 4$s with a sampling time of $T_s = 0.025$s (i.e. 160 simulation steps and corresponding search trees), a time interval sufficient to stabilize the pendulum starting from any state. Online tests compare the returns (cumulative rewards) obtained by the algorithms.

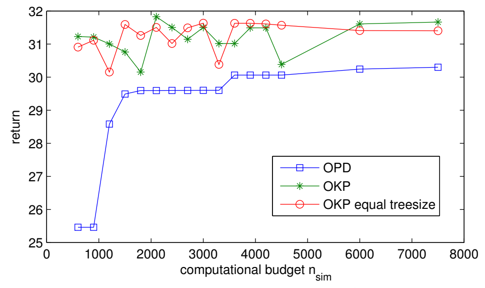

Fig. 8. Returns obtained for $K = 16$.

Looking at Fig. 8, a near-optimal return is obtained by OPD for $n_{sim} = 1500$, while the OKP variants reach the same value already for $n_{sim} = 600$. The low return obtained by OPD for $n_{sim} = 600$ indicates a suboptimal solution, with one more swing than in case of the near-optimal solutions. Hence, for certain control problems, OKP is able to obtain the near-optimal solution with smaller budget due to the use of repeated actions and the better performance is maintained for larger values of n_{sim} as well. This confirms that the problem prefers longer action sequences of the same value, for which OKP is able to find better solutions than OPD is able to.

V. Conclusions

This paper presented a newly developed control algorithm called OKP that aims to improve the performance of OPD for a subclass of control problems preferring long ranges of constant control actions. With each refinement step of the search for action sequences, OKP looks for repeated constant actions as well. By doing so, if the parameter of recurrence K is properly chosen, OKP can perform better than OPD.

While an asymptotic analysis was not forthcoming, an a posteriori guarantee has been obtained, bounding the regret based on the cumulative depth of the tree, expecting this depth to be better than the one of OPD. However, experiments have shown that this cumulative depth of the OKP search tree is in most cases smaller than the depth of the OPD tree, and therefore OKP ensures a weaker regret bound than OPD. So in this sense, our paper presents a negative theoretical result. Nevertheless, from a practical standpoint, OKP outperforms OPD in our experiments and it is expected that this conclusion can be drawn generally for a given class of problems. Future work may focus on characterizing this class, exploiting other variants of the algorithm and continuing the analysis.

References

[1] 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.

[2] Y.-Y. Cao, L. Hu, and P. Frank, “Model predictive control via piecewise constant output feedback for multirate sampled-data systems,” in *Decision and Control, 2000. Proceedings of the 39th IEEE Conference on*, vol. 1. IEEE, 2000, pp. 650–655.

[3] R. Vinter, *Optimal control*. Springer Science+ Business Media, 2010.

[4] 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.

[5] L. Busoniu, R. Munos *et al.*, “Optimistic planning for markov decision processes,” in *Proceedings of the 15th International Conference on Artificial Intelligence and Statistics, AISTATS-12*, vol. 22.

[6] 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.

[7] A. Weinstein and M. L. Littman, “Bandit-based planning and learning in continuous-action markov decision processes,” in *Proceedings of the 22nd International Conference on Automated Planning and Scheduling (ICAPS)*, 2012.

[8] M. Quincampoix and N. Seube, “Stabilization of uncertain control systems through piecewise constant feedback,” *Journal of mathematical analysis and applications*, vol. 218, no. 1, pp. 240–255, 1998.

[9] L. Magni and R. Scattolini, “Model predictive control of continuous-time nonlinear systems with piecewise constant control,” *Automatic Control, IEEE Transactions on*, vol. 49, no. 6, pp. 900–906, 2004.

[10] ——, “Tracking of non-square nonlinear continuous time systems with piecewise constant model predictive control,” *Journal of Process Control*, vol. 17, no. 8, pp. 631–640, 2007.

[11] R. Findeisen and F. Allgöwer, “Computational delay in nonlinear model predictive control,” in *Proceedings International Symposium on the Advadnced Control of Chemical Processes*, 2004, pp. 427–432.

[12] X. Yang and L. T. Biegler, “Advanced-multi-step nonlinear model predictive control,” *Journal of Process Control*, vol. 23, 2013.

[13] C. Liu, W.-H. Chen, and J. Andrews, “Piecewise constant model predictive control for autonomous helicopters,” *Robotics and Autonomous Systems*, vol. 59, no. 7, pp. 571–579, 2011.