Delft University of Technology

Delft Center for Systems and Control

Technical report 14-024

Optimistic planning with a limited number of action switches for near-optimal nonlinear control*

K. Máthé, L. Buşoniu, R. Munos, and B. De Schutter

If you want to cite this report, please use the following reference instead:

K. Máthé, L. Buşoniu, R. Munos, and B. De Schutter, "Optimistic planning with a limited number of action switches for near-optimal nonlinear control," *Proceedings of the 53rd IEEE Conference on Decision and Control*, Los Angeles, California, pp. 3518–3523, Dec. 2014. doi:10.1109/CDC.2014.7039935

Delft Center for Systems and Control
Delft University of Technology
Mekelweg 2, 2628 CD Delft
The Netherlands
phone: +31-15-278.24.73 (secretary)
URL: https://www.dcsc.tudelft.nl

* This report can also be downloaded via https://pub.bartdeschutter.org/abs/14_024

Optimistic Planning with a Limited Number of Action Switches for Near-Optimal Nonlinear Control

Koppány Máthé Lucian Buşoniu Rémi Munos Bart De Schutter

***Abstract*— We consider infinite-horizon optimal control of nonlinear systems where the actions (inputs) are discrete. With the goal of limiting computations, we introduce a search algorithm for action sequences constrained to switch at most a given number of times between different actions. The new algorithm belongs to the *optimistic planning* class originating in artificial intelligence, and is called *optimistic switch-limited planning* (OSP). It inherits the generality of the OP class, so it works for nonlinear, nonsmooth systems with nonquadratic costs. We develop analysis showing that the switch constraint leads to polynomial complexity in the search horizon, in contrast to the exponential complexity of state-of-the-art OP; and to a correspondingly faster convergence. The degree of the polynomial varies with the problem and is a meaningful measure for the difficulty of solving it. We study this degree in two representative, opposite cases. In simulations we first apply OSP to a problem where limited-switch sequences are near-optimal, and then in a networked control setting where the switch constraint must be satisfied in closed loop.**

I. Introduction

Optimal control problems arise in numerous areas of technology. We focus here on optimal control in discrete time, so as to maximize a discounted sum of rewards (negative costs). Recently proposed in artificial intelligence, the class of *optimistic planning* (OP) techniques [17] solves the optimal control problem locally at any given state, by exploring tree representations of possible sequences of actions (inputs) from that state. Given a computational budget of tree node expansions, performance grows with the resulting depth of the tree (an adaptive horizon). OP works for general dynamics and rewards, and has the crucial advantage of providing a tight characterization of the relation between the computational budget and near-optimality. Motivated by these features, several OP algorithms have been introduced, e.g. [6], [12], [15], and they have shown good performance in practical problems [10], [15]. OP can be applied offline to find arbitrarily long near-optimal sequences, but it is usually applied online in a receding-horizon fashion, so that it is a type of model-predictive control (MPC).

In this paper, we consider deterministic systems with discretized actions, and introduce a new OP technique tailored for sequences that switch only a limited number of times between different discrete actions. The constraint on the number of switches is exploited by only exploring sequences that switch at most S times, resulting in an algorithm we call *Optimistic Switch-limited Planning* (OSP). This allows us to significantly reduce the computational complexity with respect to the state-of-the art algorithm in the deterministic, discrete-action case: Optimistic Planning for Deterministic systems (OPD) [11]. OSP inherits the generality of OP, so it can deal with generic, nonlinear and nonsmooth dynamics and nonquadratic, nondifferentiable reward functions.

The switch constraint is motivated by two classes of problems. In the first class (i), the loss of performance induced by the constraint is negligible, e.g. in time-optimal control, where solutions are of the bang-bang type. In the second class (ii), the constraint must be imposed due to the problem's nature, despite performance degradation, e.g. because setting the actuator to a new discrete level is costly, or simply to decrease computational efforts. Examples include traffic signal control [9], water level control by barriers and sluices [21], networked control systems [19], etc.

The idea of exploiting limited switches is novel in OP. In MPC, exploiting such a constraint to decrease computation has been proposed in the linear case in [1], [8], [9], later extended to the nonlinear case as time-instant optimization MPC [21], and with applications to hybrid [1], [9], [16] or hierarchical control [18]. In [14], the solutions are constrained to hold the command constant for a preset number of steps. In these works, an off-the-shelf optimizer (e.g. of the mixed-integer linear programming type [2]) is usually applied, and computation is investigated empirically. In contrast, our approach analytically characterizes the relationship between computation and near-optimality, for the complete algorithm down to the implementation of the optimizer.

Indeed, by exploiting the switch constraint we show that the complexity of OSP when applied at any state is polynomial in the tree depth, rather than exponential as in OPD. Therefore, given a computation budget n, the tree depth grows quickly and (since near-optimality is related to depth) OSP converges faster to the *constrained* optimal solution than OPD would converge to the *unconstrained* one. More precisely, we introduce an asymptotic measure of problem complexity at the given state called near-optimality degree, σ, so that the size of the tree that OSP must explore to reach depth d is $O(d^\sigma)$.[1] Then, d grows like $n^{1/\sigma}$ and suboptimality decreases exponentially with this

K. Máthé (koppany.mathe@aut.utcluj.ro) and L. Buşoniu are with the Automation Department, Technical University of Cluj-Napoca, Romania. R. Munos is with INRIA Lille, France. B. De Schutter is with the Delft Center for Systems and Control, Delft University of Technology, the Netherlands. This paper is supported by the Sectoral Operational Programme Human Resources Development (SOP HRD), ID 137516 financed from the European Social Fund and by the Romanian Government; by a grant of the Romanian National Authority for Scientific Research, CNCS-UEFISCDI, project number PNII-RU-TE-2012-3-0040; and by an internal grant of the Technical University of Cluj-Napoca.

We are grateful to Anna Sadowska and Bart Dekens for helpful discussions on the examples.

[1] Here and in the sequel, notations $g = O(h)$ and $g = \Omega(h)$ mean that g asymptotically grows, respectively, at most as fast / at least as fast as h.

depth. In simpler problems, σ is smaller, so that d grows more quickly and suboptimality decreases faster. The near-optimality degree σ is related to other complexity measures in OP, e.g. the near-optimality exponent [6] or the branching factor of near-optimal sequences [11]. However, the structure of the OSP tree is more specific, so these measures would not work here.

The method is applied in receding horizon to the inverted pendulum swingup, which is in class (i) where near-optimal sequences switch rarely. In addition, we show how the switch constraint can be applied to enforce bandwidth limitations in networked control systems, leading to a problem from class (ii). The constraint is enforced in closed loop so that along any range of N consecutive steps there are at most S switches, where N is a parameter. For (i) the *closed-loop* solution may still switch often. We also explain how to reuse the developed trees across multiple steps.

Other work in MPC that characterizes complexity typically focuses on the linear quadratic case, e.g. [13], with a particularly strong work thread in explicit MPC [4], where the optimal state feedback law is piecewise affine and the complexity of the online search for the current affine region is characterized, see e.g. [3], [20]. Recall that OSP works for nonlinear dynamics and nonquadratic reward functions.

Next, Section II gives the necessary background, Section III introduces and analyzes the proposed approach, and Section IV evaluates it in examples. Section V concludes.

II. Background: Optimal Control and Optimistic Planning for Deterministic Systems

Consider a Markov decision process (MDP) describing an optimal control problem with state $x \in X$, action $u \in U$, transition function $f : X \times U \to X$ and an associated reward function $\rho : X \times U \to \mathbb{R}$. Function f describes the transition from state x to next state x' when applying action u, i.e. the system dynamics, $x' = f(x, u)$. The immediate quality of the transition is rewarded by $\rho(x, u)$.

We assume that the action space U is finite and discrete, $U = \{u^1, ..., u^M\}$, and the system dynamics f and the reward function ρ are known. Additionally, the reward function is considered to be bounded, $\rho(x, u) \in [0, 1], \forall x, u$.[2]

The objective is to find for any given state x_0 an infinite action sequence $h_\infty = (u_0, u_1, ...)$ that maximizes the value function (discounted sum of rewards)

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

where $\gamma \in [0, 1)$ is the discount factor. The optimal value is denoted $v^* = \sup_{h_\infty} v(h_\infty)$.

The Optimistic Planning for Deterministic Systems (OPD) algorithm [11], [17] is an extension of the classical A^* tree search to infinite-horizon problems. OPD looks for v^* by creating a search tree starting from x_0, and simulating action sequences until a given computational budget is exhausted. At the end, it chooses an action sequence that maximizes the discounted sum of rewards of sequences on the tree. Usually, to deal with unmodeled effects, only the first action of this sequence is applied to the system, after which the loop is closed and the algorithm is reapplied for the new state.

We define the computational budget n as the number of nodes the algorithm is allowed to expand in the search tree, where expanding a node means adding M child nodes to it, one corresponding to each action from U. The example from Fig. 1 therefore considers $n = 3$ expansions.

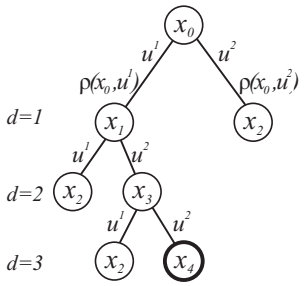

Fig. 1. OPD search tree with states in the nodes and actions and rewards on the arcs (corresponding to transitions). The size of the action space is $M = 2$. A node at depth d can also be seen as the action sequence h_d that leads to that state starting from the root state: one example at $d = 3$ is (u^1, u^2, u^2). Note that some states may appear several times in the tree as results of different action sequences. The algorithm does not exploit these duplicates, but keeps them separate on the tree.

The search tree is constructed by iteratively expanding the optimistic leaf, defined next. Considering a node at depth d as equivalent to the action sequence $h_d = (u_0, u_1, ..., u_{d-1})$ leading to it, the optimistic leaf is the node with the highest *b*-value, where a *b*-value is an upper bound defined on $v(h_\infty)$ for all the sequences h_∞ passing through a node h_d:

$$b(h_d) = \nu(h_d) + \frac{\gamma^d}{1-\gamma} \geq v(h_\infty) \tag{2}$$

where $\nu(h_d)$ defines a lower bound on $v(h_\infty)$:

$$\nu(h_d) = \sum_{k=0}^{d-1} \gamma^k \rho(x_k, u_k) \leq v(h_\infty) \tag{3}$$

These are valid bounds on $v(h_\infty)$ since $\gamma < 1$, and $\rho(x_k, u_k) \in [0, 1]$. Note that since the depth d varies among leaf nodes, so does the gap $\frac{\gamma^d}{1-\gamma}$ between $\nu(h_d)$ and $b(h_d)$, and therefore $b(h_d)$ must be computed separately and constitutes an informative selection criterion.

OSP returns a sequence maximizing the lower bound:

$$\hat{h}^* = \arg \max_{\text{leaf } h_d} \nu(h_d) \tag{4}$$

III. Optimistic Switch-Limited Planning

The OSP algorithm is based on the same principle as OPD: it simulates action sequences starting from state x_0 by optimistically constructing a search tree. After the simulation ends, like OPD, OSP chooses a sequence $\hat{h}^*$ maximizing $\nu(h_d)$, see (4). The novelty of OSP is a constraint applied to the algorithm: it never expands a node with more than S action switches in its action sequence. Some examples of finite action sequences respecting this constraint are: for $S = 1$, $h_3 = (u^2, u^1, u^1)$, for $S = 2$, $h_3 = (u^2, u^1, u^1)$ or $h_5 = (u^2, u^3, u^3, u^1, u^1)$. An example of an OSP search

[2] If the reward function is bounded in any other interval, it can be scaled and translated in $[0, 1]$ without affecting the optimal solution.

tree, showing all sequences created by OSP up to depth 4, is presented in Fig. 2.

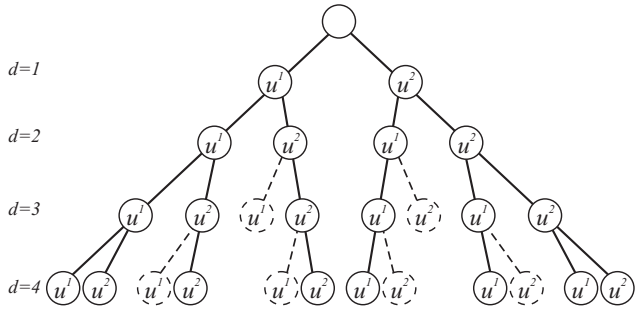

Fig. 2. OSP tree up to depth 4, with $M = 2$ discrete actions and $S = 1$ switch allowed. The action leading to a node is put in the node. Nodes preceded by continuous lines are expanded with all M actions. Nodes on the dashed paths are no longer expanded, since they have more than S switches in their action sequence.

We fully analyze OSP when applied locally at a given state x_0, and later show how to empirically apply it in closed loop. Using the constraint on the number of switches, the search space is restricted and the optimal solution may fall outside it. Denote the optimal return with sequences having at most S switches by v_S^*, where $v_S^* \leq v^*$. In problem class (i) (see Section I), the loss $v^* - v_S^*$ due to enforcing the switch limitation is small by assumption, whereas in class (ii) it may be significant but it must be accepted due to the problem constraints or the need to reduce computational efforts. As $S \to \infty$, v_S^* is expected to approach v^*.

Intuitively, the constraint allows OSP to construct deeper search trees than OPD for a given problem, as in place of the nodes eliminated by the constraint OSP will explore other nodes that may be at larger depths. Therefore, for the same budget n, OSP ensures a smaller distance to the constrained optimum v_S^* than OPD would ensure with respect to v^*.

To make the following statements more concise, we say that OSP is ε-optimal if the solution $\hat{h}^*$ it returns satisfies $v_S^* - \nu(\hat{h}^*) \leq \varepsilon$.

Lemma 1. OSP will only expand nodes that satisfy the relation $v_S^* - \nu(h_d) \leq \frac{\gamma^d}{1-\gamma}$. Therefore, the algorithm is $\frac{\gamma^{d_{\max}}}{1-\gamma}$-optimal, where $d_{\max}$ is the depth of the deepest expanded node, information available a posteriori.

Proof. Define the set $\mathcal{H}_d$ consisting of all infinite sequences h_∞ starting with h_d; then, there exists at least one leaf h_d in a search tree such that $\sup_{h_\infty \in \mathcal{H}_d} v(h_\infty) = v_S^*$. Therefore, using the definition of the *b*-value (2), there exists h_d such that $b(h_d) \geq v_S^*$. Since the algorithm always expands the node with the highest *b*-value, all the expanded nodes will satisfy the relation $b(h_d) \geq v_S^*$. Hence, $v_S^* - \nu(h_d) \leq \frac{\gamma^d}{1-\gamma}$ is valid for all the expanded nodes.

Now, as $\nu(\hat{h}^*) \geq \nu(h_{d_{\max}})$ due to (4), and $b(h_{d_{\max}}) = \nu(h_{d_{\max}}) + \frac{\gamma^{d_{\max}}}{1-\gamma} \geq v_S^*$ from the previous part of the proof, we get $v_S^* - \nu(\hat{h}^*) \leq \frac{\gamma^{d_{\max}}}{1-\gamma}$, and the proof is complete. ■

This result is similar to the one for OPD (where v_S^* would be substituted by v^*), which was given in a different form in [11]. Note that $\hat{h}^*$ may be among the nodes with an extra switch, which are never expanded and so remain leaf nodes; the entire analysis holds in that case as well. (The way to apply OSP in case (ii), which will be explained in Section III-C, further ensures that selecting such a node does not affect the satisfaction of the constraint in closed loop.)

Now, note that due to Lemma 1 and the switch constraint, at depths up to d OSP only expands nodes in the set:

$$H_d = \left\{ h_{d'} |\ d' \leq d; s(h_{d'}) \leq S; v_S^* - \nu(h_{d'}) \leq \frac{\gamma^{d'}}{1-\gamma} \right\}$$

where $s(h_{d'})$ counts the number of switches in the action sequence $h_{d'}$. Using the cardinality of this set one can characterize a priori the depth the algorithm will reach.

In OPD, the nodes possibly expanded at depth d do not have to satisfy the switch limitation, but only be $\frac{\gamma^d}{1-\gamma}$-optimal. It was shown then in [11] that the tree of such nodes grows with branching factor $K \in [1, M]$, a measure of complexity of the OPD planning problem. Then, the number of expandable nodes grows exponentially with the depth, so that the cardinality of the entire tree up to depth d is dominated by the number of nodes at this last depth: it is $O(K^d)$. In OSP this is no longer valid, as the search tree grows polynomially with the depth. To see why, consider the worst possible case, in which nodes are expanded in the order of their depth, see also Section III-A. Then, OSP distributes up to S switches along sequences of length d, a number proportional to the combinations of up to S elements from d – which grows only polynomially with d.

Now, to describe *in general* the cardinality of H_d, a new complexity measure is needed, and is defined as follows.

Definition. Let $c > 0$ and $\sigma \in [1, S+1]$ be so that $|H_d| \leq c \cdot d^\sigma$; we show later that these constants always exist. The *near-optimality degree* is defined as the smallest value of σ for which the relationship holds.

Our results below hold for any pair c, σ, but we take a pair with the smallest σ, which is called near-optimality degree because it plays a similar role to the near-optimality dimension from the optimization algorithms at the basis of OP [17]. Note that σ (as well as K in OPD) may be non-integer.

Theorem 2. Given a computational budget n, the OSP algorithm is $\frac{\gamma^{(n/c)^{1/\sigma}}}{1-\gamma}$-optimal, where c is the constant from the definition of σ.

Proof. According to Lemma 1, the OSP algorithm is $\frac{\gamma^{d_{\max}}}{1-\gamma}$-optimal if a node at depth $d_{\max}$ was expanded. Now, we calculate a lower bound on $d_{\max}$ a priori from the cardinality of set H_d. Define d^* to be the smallest depth so that $n \leq |H_{d^*}| = c \cdot d^{*\sigma}$; this means the algorithm has surely expanded nodes at d^* (but possibly not yet at d^*+1), so $d_{\max} \geq d^*$. Moreover, $d^* \geq (n/c)^{1/\sigma}$, hence the same holds for $d_{\max}$, and OSP is $\frac{\gamma^{(n/c)^{1/\sigma}}}{1-\gamma}$-optimal. ■

A smaller σ corresponds to a slower growth of H_d, so that a given budget n allows reaching larger depths and thus a better solution. In particular, in the best case $\sigma = 1$, which means that H_d grows linearly with d, and suboptimality shrinks exponentially with n. In the sequel, this theorem and the meaning of σ are illustrated for two interesting opposite cases, omitting the detailed proofs due to limited space.

A. Identical rewards ($\sigma = S + 1$)

Consider a problem where all the rewards are identical, say equal to 1 or to 0. While any sequence is optimal in this problem, it is nevertheless an interesting worst case, which highlights the (correct) behavior of the algorithm in general, as we explain below.

Proposition 3. In case of identical rewards, $\sigma = S + 1$ and the OSP algorithm is $\frac{\gamma^{\frac{1}{M}(n/c')^{1/(S+1)}}}{1-\gamma}$-optimal, where c' is a positive constant.

Proof sketch. Each node at depth d corresponds to a way of choosing s switches among d steps and M actions, overall $\binom{dM}{s} \leq (Md)^s$. Counting all $s \leq S$, nodes at lower depths, and other factors, it takes $n = O((Md)^{S+1})$ expansions to reach depth d, and the result follows by Lemma 1. ■

This bound differs from the one of Theorem 2 by including M, the number of actions, which yields a more precise expression and a different constant c'; if this refinement were ignored, we would obtain the general result of Theorem 2.

So OSP expands nodes uniformly, in the order of their depth, and σ has the largest possible value. The bound achieved by OSP here is also the smallest achievable in a worst-case sense, which means that for any planning algorithm, and any value of S and n, one can find a problem (constrained to sequences with at most S switches) for which the distance from the optimal value is $\Omega(\gamma^{\frac{1}{M}(n/c')^{1/(S+1)}})$. To see this, choose the largest D so that $n \geq c'[M(D-1)]^{(S+1)}$, assign rewards of 1 for some arbitrary sequence h^*_∞ satisfying the constraint, but only starting from level $D+1$ onward, and rewards of 0 everywhere else. Then, OSP has uniformly expanded all nodes up to $D-1$ but none at $D+1$, so it has no information and must make an arbitrary action choice, which may not be optimal, leading to a sub-optimality of $\frac{\gamma^{D+1}}{1-\gamma} = \Omega(\gamma^{\frac{1}{M}(n/c')^{1/(S+1)}})$. An algorithm that does not expand uniformly may miss the optimal sequence for an even larger number of expansions n, so their suboptimality is at least as large. This fact also shows that OSP *behaves correctly*: as long as only uniform rewards are observed, the tree must be expanded uniformly, and this behavior is reflected in the bound.

B. Single optimal path ($\sigma = 1$)

In this case, a single sequence has maximal rewards (equal to 1), and all other transitions have a reward of 0.

Proposition 4. When there is a single optimal path, $\sigma = 1$ and OSP is $\frac{\gamma^{n-c''}}{1-\gamma}$-optimal, with c'' a positive constant.

Proof sketch. When the optimal path has at most S switches, OSP only explores this path. Otherwise, at some depth it reaches the switch limit and then continues expanding the subsequent constant-action path (a constrained optimal solution), while for each node expanded along this path it may also expand a constant number of nodes elsewhere. In both cases, $n = O(d)$ leading to the bound. ■

Thus, in this case where a "maximal" amount of structure exists in the reward function, the problem becomes easy, represented by a value of $\sigma = 1$, and the near-optimality is exponential in n rather than stretched-exponential like before. Again, we have obtained a refined expression with a different constant than in Theorem 2.

These special cases also prove the existence of $\sigma \in [1, S+1]$, as the tree cannot contain more nodes than in the case of uniform expansion (thus $\sigma \leq S + 1$), nor fewer nodes than in the single-optimal-path case (thus $\sigma \geq 1$).

C. Implementation in closed loop

To implement the algorithm in closed loop, one simply applies it at each encountered state x_k, sends the first action of the sequence returned to the actuator, and then repeats the process in receding horizon. In case (i), the resulting closed-loop sequence may have more than S switches.

Consider now case (ii), when the switch constraint must be enforced in closed loop. It is not practical to enforce only S switches for the entire infinite horizon, as after they are exhausted the algorithm can no longer react to unmodeled effects (e.g. disturbances). Instead, we suggest ensuring at most S switches are applied for any range of consecutive N steps, where N is a tuning parameter. This can be implemented easily, by keeping track of the past N actions (over the range $k - N, \ldots, k - 1$) and ensuring that nodes violating the condition are not expanded by the algorithm. Further, for the ranges of steps where no more switches are allowed, the action is simply held constant and OSP is only reapplied when a new switch becomes eligible.

Finally, we explain how the tree developed at step k can be used as a starting point at $k+1$. We start with the subtree corresponding to the single action applied, and throw away the subtrees of the other actions. Since all sequences on this subtree are truncations of sequences on the original tree, all expanded nodes satisfy the constraint, while new nodes become eligible for expansion – which the algorithm can proceed to expand. Thus, performance at $k+1$ increases by reusing the subtree. This holds in case (i), as well as when the closed-loop constraint from case (ii) is enforced.

IV. Simulation Results

OSP is first evaluated in the problem of swinging up an inverted pendulum (Section IV-A). Although an academic example, this is an appropriate benchmark for our algorithm, allowing us to study it in detail: the example is nonlinear, requires large horizons to plan the swings, and the swings exhibit the limited-switch property so the problem is in class (i). Then, as a more general set of applications, we show how the constraint from case (ii) can help in networked control systems, and illustrate the idea using again the inverted pendulum (Section IV-B).

In all simulations, we apply only the first action from the action sequence an algorithm returns. We compare OSP against OPD using various performance indicators, for a range of values of the number of expansions allowed n.

A. Inverted Pendulum Swing-up

The goal in the Inverted Pendulum problem is to bring and stabilize a pendulum to the pointing-up equilibrium, starting from any state $x = [x_1,\ x_2]^T$, with $x_1 = \alpha \in [-\pi, \pi)$ rad

the angular position and $x_2 = \dot{\alpha}$ [rad/s] the angular velocity of the pendulum; the latter is restricted to $[-15\pi, 15\pi]$ rad/s, by saturation. The pendulum is controlled using a motor as shown in Fig. 3, with the control action space $U = \{-2, 0, 2\}$ V. 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.

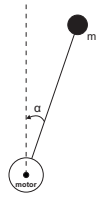

Fig. 3. Inverted Pendulum

The first set of simulations considers with each value of n a set of states $\left\{-\pi, -\frac{5\pi}{6}, ..., \pi\right\}\text{rad} \times \{-15\pi, -14\pi, ..., 15\pi\}\,\text{rad/s}$ for which the algorithms are run, providing a near-optimal control action for each state. Fig. 4 presents the average regret, while Fig. 5 shows the depth of the deepest expanded nodes, averaged over this set of states, for a range of values of n. The regret is difference between the unconstrained optimal value v^* and the value $v(u_0)$ of applying the first action in the returned sequence, and then acting optimally; so lower regret is better. The regret measure is appropriate for a receding-horizon algorithm. In this problem, the true optimum v^* is not available but we compute an accurate approximation of it using value iteration with a precise approximator [5], at much higher computational costs.

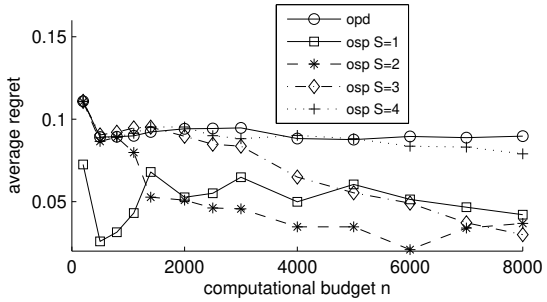

Fig. 4. Inverted Pendulum average regret for OPD and for OSP with $S \in \{1, 2, 3, 4\}$.

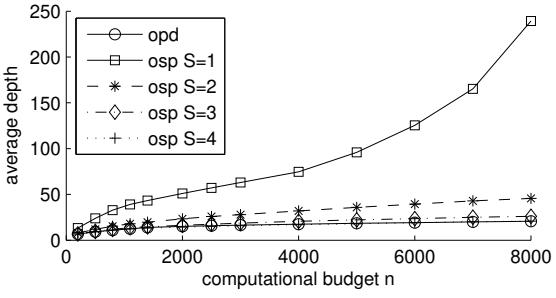

Fig. 5. Inverted Pendulum average depth.

A first remark is that average regret tends to decrease with n, i.e. with more expansions OSP generally obtains better results, but the relationship is not strict. The depth on the other hand always increases when allowing more expansions. Recall from Lemma 1 that as depth grows suboptimality is reduced, however for OSP the relation is not exactly reflected by the two figures since we measure regret in the unconstrained problem.

The advantage of OSP over OPD can be clearly seen for $S \leq 3$. For the same budget, OSP gets closer to the constrained optimum, which here is also close to the true optimum v^* and so the overall regret is better. Also note that as S increases, OSP converges to OPD as expected from the design, a property of the algorithm reflected in the following simulations as well.

The next set of simulations consists of running the algorithms online, in receding horizon from initial state $x_0 = [-\pi, 0]^T$ and for 160 steps, with a sampling time of 0.025s. For the resulting finite sequence of control actions the return (1) is calculated. The following figures show the returns obtained by the algorithms for the same range of n as before. Note that the online simulations result in different sets of 160 states (due to different trajectories) for each value of the computational budget, different from the previous simulations as well; so these returns cannot be directly compared to the average-regret results before.

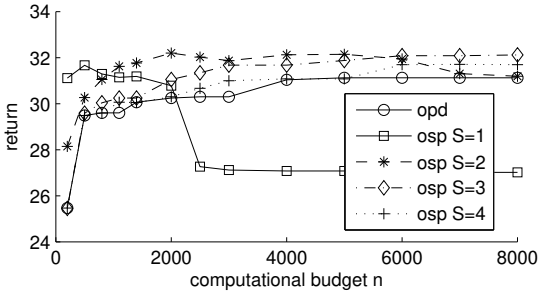

Fig. 6. Inverted Pendulum return for OPD and OSP.

Looking at Fig. 6, a first remark is that the convergence of OSP to OPD with $S \to \infty$ is reflected here as well.

Choosing $S = 1$ is insufficient and OSP obtains a suboptimal solution (the larger returns obtained for small n OSP are a coincidence). Taking higher values for S, the advantage of OSP is clearly reflected: the return increases with n, while obtaining the same return as OPD for lower values of n. In other words, for intermediate values of S, OSP obtains a result of the same quality as OPD, using fewer computational resources (as also illustrated by the regret, before).

Regarding computation time, for the same budget OSP runs somewhat faster than OPD, as it selects nodes to expand by looking at candidates in a constrained (i.e. smaller) set of leaves. We omit detailed results due to space limits.

B. Application to networked control systems

In networked control systems, the controller is connected to the system by a communication network shared with other devices (e.g. controllers). We consider the setting where state measurements can be performed at every step, while changes in the control action are expensive and should be minimized; this is standard in so-called event-triggered control [19].

Then, a new action is transmitted only when a switch occurs, and in the meantime the old action is maintained. By setting the ratio of S switches (transmissions) per N steps, OSP can be used to fine-tune the usage of the network. Discretizing the actions in a small number of actions is also useful [7], since it reduces the size of the packets to a few bits (the action index), requiring a local table to transform the action index back into the true value on the system side. Fig. 7 shows this architecture.

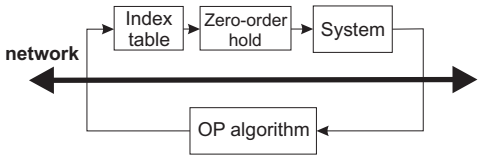

Fig. 7. Networked control system architecture

We illustrate this setting for the inverted pendulum. OSP is used to enforce at most $S = 3$ action changes over $N = 12$ or 20 steps (as the sampling time is 0.025s and $M = 3$, no more than 9 bits of data are sent in 0.3s or 0.5s, respectively). As expected from the previous section, in Fig. 8 OSP is still able to maintain a better performance than OPD despite the restriction on the number of switches. Note that OPD violates the constraint, by switching e.g. 4 times over 10 steps.

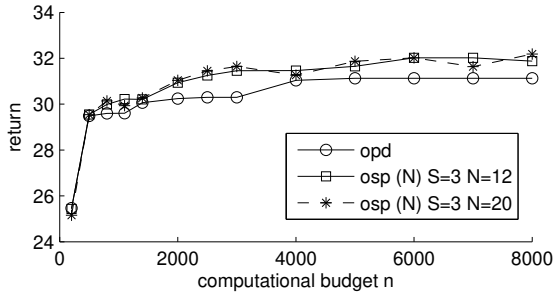

Fig. 8. Returns of OSP when the closed-loop switch constraint is enforced, compared to OPD.

V. Conclusions and Future Work

This paper has presented a novel algorithm for near-optimal control called OSP, tailored to compute solutions that switch at most S times between different discrete actions. By taking advantage of this property, our analysis shows that computation is polynomial in the adaptive horizon of the computed solution, in contrast to the exponential complexity achieved by unconstrained OP algorithms; and therefore that OSP approaches the constrained optimum fast. The degree of the polynomial, called *near-optimality degree*, characterizes the difficulty of the optimal control problem at a given state, with smaller degrees corresponding to simpler problems. Extensive simulations confirm the analytical properties of the algorithm and illustrate that it works well in receding horizon. As a more general class of applications, we explain how the switch constraint can be applied to limit bandwidth requirements in networked control systems.

Future work will focus on analyzing the closed-loop performance, and on developing an adaptive-S variant of the algorithm that should converge fast to the *unconstrained* optimal solution.

References

[1] J. Alende, Y. Li, and M. Cantoni, "A {0,1} linear program for fixed-profile load scheduling and demand management in automated irrigation channels," in *Proceedings 48th IEEE Conference on Decision and Control (CDC-09)*, Shanghai, China, 16–18 December 2009, pp. 597–602.

[2] M. J. Alves and J. Clímaco, "A review of interactive methods for multiobjective integer and mixed-integer programming," *European Journal of Operational Research*, vol. 180, no. 1, pp. 99–115, 2007.

[3] F. Bayat, T. A. Johansen, and A. A. Jalali, "Using hash tables to manage the time-storage complexity in a point location problem: Application to explicit model predictive control," *Automatica*, vol. 47, no. 3, pp. 571–577, 2011.

[4] A. Bemporad, M. Morari, V. Dua, and E. N. Pistikopoulos, "The explicit linear quadratic regulator for constrained systems," *Automatica*, vol. 38, no. 1, pp. 3–20, 2002.

[5] 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.

[6] 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.

[7] C. De Persis and P. Frasca, "Robust self-triggered coordination with ternary controllers," *IEEE Trans. Automat. Contr.*, vol. 58, no. 12, pp. 3024–3038, 2013.

[8] B. De Schutter, "Optimal control of a class of linear hybrid systems with saturation," *SIAM Journal on Control and Optimization*, vol. 39, no. 3, pp. 835–851, 2000.

[9] B. De Schutter and B. De Moor, "Optimal traffic light control for a single intersection," *European Journal of Control*, vol. 4, no. 3, pp. 260–276, 1998.

[10] S. Gelly, Y. Wang, R. Munos, and O. Teytaud, "Modification of UCT with patterns in Monte-Carlo Go," INRIA, Paris-Sud, France, Tech. Rep., 2006.

[11] 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.

[12] 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.

[13] X. Li and T. E. Marlin, "Model predictive control with robust feasibility," *Journal of Process Control*, vol. 21, no. 3, pp. 415–435, 2011.

[14] 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.

[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] C. O. Martinez, A. Bemporad, A. Ingimundarson, and V. P. Cayuela, "On hybrid model predictive control of sewer networks," in *Identification and Control*, R. Sánchez, V. Puig, and J. Quevedo, Eds. Springer London, 2007, pp. pp 87–114.

[17] R. Munos, "The optimistic principle applied to games, optimization and planning: Towards foundations of Monte-Carlo tree search," *Foundations and Trends in Machine Learning*, vol. 7, no. 1, pp. 1–130, 2014.

[18] A. Sadowska, B. De Schutter, and P.-J. van Overloop, "Event-driven hierarchical control of irrigation canals," in *Proceedings of the US-CID Seventh International Conference on Irrigation and Drainage*, Phoenix, Arizona, Apr. 2013.

[19] P. Tabuada, "Event-triggered real-time scheduling of stabilizing control tasks," *IEEE Transactions on Automatic Control*, vol. 52, no. 9, pp. 1680–1685, 2007.

[20] P. Tøndel, T. A. Johansen, and A. Bemporad, "Evaluation of piecewise affine control via binary search tree," *Automatica*, vol. 39, no. 5, pp. 945–950, 2003.

[21] H. van Ekeren, R. Negenborn, P. van Overloop, and B. De Schutter, "Time-instant optimization for hybrid model predictive control of the Rhine-Meuse delta." *Journal of Hydroinformatics*, vol. 15, no. 2, pp. 271–292, 2013.