Direct model predictive control

Jean-Joseph Christophe, Jérémie Decock, Olivier Teytaud

Direct model predictive control

Jean-Joseph Christophe and Jérémie Decock and Olivier Teytaud

TAO (Inria), LRI, UMR 8623 (CNRS - Univ. Paris-Sud),
bat 490 Univ. Paris-Sud 91405 Orsay, France, firstname.lastname@inria.fr

Abstract. Due to simplicity and convenience, Model Predictive Control, which consists in optimizing future decisions based on a pessimistic deterministic forecast of the random processes, is one of the main tools for stochastic control. Yet, it suffers from a large computation time, unless the tactical horizon (i.e. the number of future time steps included in the optimization) is strongly reduced, and lack of real stochasticity handling. We here propose a combination between Model Predictive Control and Direct Policy Search.

1 Introduction

Since the early times of stochastic dynamic optimization, power systems are a great challenge, due to large scale constrained decision spaces, high stochasticity and dozens of state variables. The traditional approaches are based on Bellman's decomposition, which is reliable, principled, well known[1]. However, in spite of significant improvements such as the use of duality[2] and careful implementation/parallelization[3], such value-based approaches have scalability issues w.r.t. the number of state variables; in particular when non-convex Bellman values or non-Markovian random processes make dual programming difficult to apply. As a consequence, Model Predictive Control and Direct Policy Search become important alternatives. We present these methods in Section 2, as well as a new combination. We then study these methods in Section 3.

2 Methods

Framework. We assume that a strategic horizon $T \in \mathbb{N}$, a state space S, an action space A, a random process $p_0, \dots, p_{T-1}$ with values $p_i \in P$, a transition function $f : S \times A \times P \to S$, a reward function $r : S \times A \to \mathbb{R}$ and an initial state $\boldsymbol{s}_0$ are given. Equipped with a policy $\pi : (\mathbb{N}, S, P) \to A$, they define actions $\boldsymbol{a}_t = \pi(t, \boldsymbol{s}_t, p_t)$, states $\boldsymbol{s}_{t+1} = f(\boldsymbol{s}_t, \boldsymbol{a}_t, p_t)$ and the total reward $R(\pi) = \mathbb{E} \sum_{t=0}^{T-1} r(\boldsymbol{s}_t, \boldsymbol{a}_t)$ for time steps $t \in \{0, \dots, T-1\}$. The goal of stochastic dynamic programming is to find (approximately) the optimum $\arg\max_\pi \mathbb{E}R(\pi)$. Bellman's decomposition[1] are classical tools for doing so when $(p_t)_{t \in \{1,\dots,T-1\}}$ is Markovian. **Model Predictive Control** (MPC[4, 5]) consists in replacing the random process by a deterministic approximation (which is usually more or

less pessimistic in order to reduce the risk). More formally:

$$\pi_{MPC}(t, \boldsymbol{s}_t, p_t) = \arg\max_{\boldsymbol{a}_t} \sup_{\boldsymbol{a}_{t+1},\ldots,\boldsymbol{a}_{T-1}} \sum_{i=t}^{T-1} r(\boldsymbol{s}_i, \boldsymbol{a}_i) \quad (1)$$

$$\forall t \leq t' < T,\ \boldsymbol{s}_{t'+1} = f(\boldsymbol{s}_{t'}, \boldsymbol{a}_{t'}, p'_{t'}),\ \text{with } p'_{t'} = \text{quantile}_q(p_{t'}),\ \forall t' > t \quad (2)$$

$$\text{and } p'_t = p_t \quad (3)$$

where quantile_q is the quantile operator, for some $q \in]0, 1[$. If the maximum in Eq. 1 is not reached, an approximation is used. **Properties of MPC:** Eq. 3 implies that constraints are satisfied. Therefore, at least, π_{MPC} is a legal policy, which is not necessarily the case for some DPS (see below). The quantile (Eq. 2) can be (i) the quantile of the random variable $p_{t'}$; we will refer to this case in the rest of this paper; (ii) the quantile of the random variable $p_{t'}|p_t$ (i.e. conditionally to the current observation p_t); in this case there is an additional cost for computing it online. Other operators than the quantile can be preferred. **Direct Policy Search** (DPS) is a rather different approach. A parametric function $(t, \boldsymbol{s}, p) \mapsto \pi_{\boldsymbol{\theta}}(t, \boldsymbol{s}, p)$, depending on a parameter $\boldsymbol{\theta}$ is defined, using general function approximators or using human expertise. A simulator is defined as follows:

Procedure 1 Simulator($\pi_{\boldsymbol{\theta}}$, $\boldsymbol{s}_0$)

Input: an initial state $\boldsymbol{s}_0$ and a policy $\pi_{\boldsymbol{\theta}}$
Output: a reward R associated to the policy $\pi_{\boldsymbol{\theta}}$
  Randomly draw $p_1, \ldots, p_{T-1}$
  for $t \in \{0, \ldots, T-1\}$ **do**
    $\boldsymbol{a}_t = \pi(t, \boldsymbol{s}_t, p_t)$
    $\boldsymbol{s}_{t+1} = f(\boldsymbol{s}_t, \boldsymbol{a}_t, p_t)$
    $r_t = r(\boldsymbol{s}_t, \boldsymbol{a}_t)$
  end for
  $R = \sum_{t=0}^{T-1} r_t$

Constraints are not necessarily satisfied by $\boldsymbol{a}_t$ chosen by $\pi(t, \boldsymbol{s}_t, p_t)$. This is a big issue for applying DPS for constrained high-dimensional DPS. Projections to the set of legal actions is possible but tricky. Then, a noisy optimization algorithm is applied for finding the optimal parameter

$$\boldsymbol{\theta}^* = \arg\max_{\boldsymbol{\theta}} \mathbb{E}\text{Simulator}(\boldsymbol{\theta}, .). \quad (4)$$

The DPS controller is by definition $\pi_{DPS} := \pi_{\boldsymbol{\theta}^*}$. A convenient DPS controller for constrained action spaces is

$$\pi_{\boldsymbol{\theta}}(t, \boldsymbol{s}_t, p_t) = \arg\max_{\boldsymbol{a}_t} r(\boldsymbol{s}_t, \boldsymbol{a}_t) + NN_{\boldsymbol{\theta}}(\boldsymbol{s}_t).f(\boldsymbol{s}_t, \boldsymbol{a}_t, p_t) \quad (5)$$

where $NN_{\boldsymbol{\theta}}$ is a neural network parametrized by $\boldsymbol{\theta}$ as discussed later. A key advantage of Eq. 5 is that contrarily to general DPS (discussed above), it ensures that constraints are satisfied (if $r(.,.)$ is infinite for constraint violations, or if the arg max is computed with constraints). **Tactical/operational horizon, MPC and DPS combination:** we define the following variants for reducing

Criterion	MPC	DPS	MPC(h)	DMPC(h)
Solving time	null	slow	null	slow
DMT	slow	fast	depends on h	depends on h
Stochasticities	deterministic	ok	deterministic	ok
Simplicity	++	-	+	–

Table 1: Comparing MPC, DPS and others. We distinguish solving time (the time for creating the controller and its parameters, from the oracle) and the decision making time DMT (the time for making a decision, during simulations, when a state and the controller with its parameters are given). Stochasticities means that the method can handle stochastic problems. Simplicity refers to the clarity and user-friendly aspects.

some weaknesses of proposed methods:

$$\pi_{MPC(h)}(t, \boldsymbol{s}_t, p_t) = \arg\max_{\boldsymbol{a}_t} \sup_{\boldsymbol{a}_{t+1},\ldots,\boldsymbol{a}_{t+h-1}} \sum_{i=t}^{t+h-1} r(\boldsymbol{s}_i, \boldsymbol{a}_i) \quad (6)$$

$$\pi_{DMPC(h,\boldsymbol{\theta})}(t, \boldsymbol{s}_t, p_t) = \arg\max_{\boldsymbol{a}_t} \sup_{\boldsymbol{a}_{t+1},\ldots,\boldsymbol{a}_{t+h-1}} \sum_{i=t}^{t+h-1} r(\boldsymbol{s}_i, \boldsymbol{a}_i) + NN_{\boldsymbol{\theta}}(\boldsymbol{s}).\boldsymbol{s}_{t+h} \quad (7)$$

$MPC(h)$ is a faster variant of MPC. h is termed the *tactical* horizon (i.e. the number of time steps considered for decision making). $DMPC(h)$ (Direct MPC) is a merge between $MPC(h)$ and DPS. Also, when the operational horizon o (number of decisions actually applied at each time step) is greater than one, we make o decisions simultaneously, as follows:

$$\pi_{MPC(h,o)}(t, \boldsymbol{s}_t, p_t) = \arg\max_{\boldsymbol{a}_{0\ldots o-1}} \sup_{\boldsymbol{a}_{o\ldots h-1}} \sum_{i=t}^{t+h-1} r(\boldsymbol{s}_i, \boldsymbol{a}_i) \quad (8)$$

$$\pi_{DMPC(h,o,\boldsymbol{\theta})}(t, \boldsymbol{s}_t, p_t) = \arg\max_{\boldsymbol{a}_{0\ldots o-1}} \sup_{\boldsymbol{a}_{o\ldots h-1}} \sum_{i=t}^{t+h-1} r(\boldsymbol{s}_i, \boldsymbol{a}_i) + NN_{\boldsymbol{\theta}}(\boldsymbol{s}).\boldsymbol{s}_{t+h} \quad (9)$$

$$\pi_{DPS}(o, \boldsymbol{\theta}) = \pi_{DMPC}(o, o, \boldsymbol{\theta}) \quad (10)$$

with $\boldsymbol{a}_{0\ldots o-1} = (a_t, \ldots, a_{t+o-1})$ and $\boldsymbol{a}_{o\ldots h-1} = (a_{t+o}, \ldots, a_{t+h-1})$. In such cases, π is applied only $\lceil T/o \rceil$ times and makes o decisions at each call. **Comparisons and summary.** We compare MPC, DPS, variants and combinations in Table 1.

3 Experiments

MPC is simple and efficient but (i) untractable on big problems with thousands of time steps; (ii) suboptimal on highly stochastic problems. MPC(h) is a classical alternative to MPC, dedicated at reducing the complexity. Our main hypothesis is that DMPC(h) is a reliable alternative to MPC and DPS.

Test bed and Experimental setup. We consider a buying/selling energy problem defined as follows: (i) 168 time steps (representing one hour each); (ii) 10 batteries; (iii) one decision per battery and per time step (the quantity of electricity to either insert or extract of each battery); (iv) perfect forecast of the energy market price at horizon 5 hours; (v) decisions made with no recourse for 5 hours (i.e. operational decisions every 5 hours in other words the *operational*

horizon is 5 hours); therefore there are $34 = \lceil 168/5 \rceil$ tuples of decisions vectors to make (tuples of h vectors in $\mathbb{R}^{10}$). We run experiments on two different cases: (i) 10 batteries vs market, normal volatility: 10 energy stocks are used for storing electricity when prices are low and selling it when prices are high; the reward is the benefit; on average over the time steps, the standard deviation of the price is 39% of the average value. (ii) 10 batteries vs market, high volatility: similar, but with a price model with larger variance; on average over the time steps, the standard deviation of the price is 62% of the average value. Thanks to this reduced test bed, we can run all methods, including the expensive MPC, which would be untractable on thousands of time steps unless a reduced tactical horizon h is used. Execution times are measured on an Intel Xeon X5660 CPU (at 2.80GHz) with 48GB RAM. We use (in MPC, DPS and DMPC) linear programming to perform the decision making faster (Eqs. 1, 5, 7). We use also for DPS and DMPC: (i) a neural network for the non-linear component of the DPS or DMPC controller (Eq. 5); (ii) an objective function based on simulations and rewards (Procedure 1);(iii) an optimization by self-adaptive evolution strategies (Eq. 4); (iv) resamplings for mitigating the noise effect. Each of these components is detailed below. The heart of fast decision making in high-dimensional context is often (polynomial time) linear programming.There are two distinct optimization problems in sequential decision making: (i) Making the decisions at a given time step, e.g. Eqs. 1, 5, 7. (ii) Optimizing the parameters of DPS, in Eq. 4. For the latter, it is difficult to make it linear. For the former, when the problem becomes large, there is no alternative for a fast decision making. MPC is usually based on linear programming, because $\boldsymbol{a} \mapsto r(\boldsymbol{s}, \boldsymbol{a})$ is linear. DPS depends on a parametric controller. When this controller is as proposed in Eqs. 1 or 7, then it boils down to linear programming; this is the key motivation for the controllers proposed in Eqs. 1 and 7. These variants of DPS make DPS tractable on large scale constrained problems. We use in this work, for DPS and DMPC, a neural network parametrized by $\boldsymbol{\theta}$, so that an additional term $NN_{\boldsymbol{\theta}}(\boldsymbol{s}).f(\boldsymbol{s}, \boldsymbol{a})$ is added (compare Eq. 8 and Eq. 9) where "." denotes the scalar product. $\boldsymbol{s} \mapsto NN_{\boldsymbol{\theta}}(\boldsymbol{s})$ is a fully connected feedforward neural network. Neural networks are general function approximators, which have been widely used in control. DPS is based on the noisy optimization $\boldsymbol{\theta}^* = \arg\max_{\boldsymbol{\theta}} \text{Simulator}(\pi_{\boldsymbol{\theta}}, \boldsymbol{s}_t)$. $\boldsymbol{\theta} \mapsto \text{Simulator}(\pi_{\boldsymbol{\theta}}, \boldsymbol{s}_t)$ is a stochastic function, hence its optimization is noisy. It is difficult, in such a case, to compute the gradient. We use a self-adaptive evolution strategy (SAES) with a resampling scheme, detailed below. SAES is a classical population-based optimization algorithm[6]. Population-based means that at each iteration, a large number of search points are evaluated independently. They are therefore highly parallelizable. We use strategies[6], with μ parent and λ offspring, where $\lambda = 4N+1$ and $\mu = N$, where N is the dimension of the search space. SAES are known for their robustness against rugged objective functions, but they suffer from noise. We here use a resampling strategy. Each search point at generation n is resampled $\lceil 10\sqrt{n} \rceil$ times, and the objective values are averaged.

Results. MPC is surprisingly efficient[5], and it easily takes into account forecasts. With pessimistic forecasts, one can mitigate the use of a deterministic

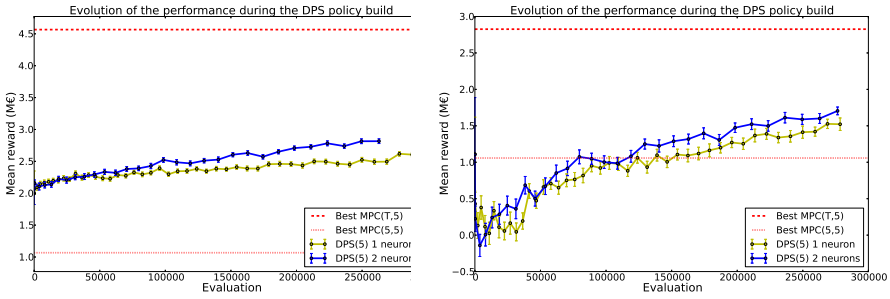
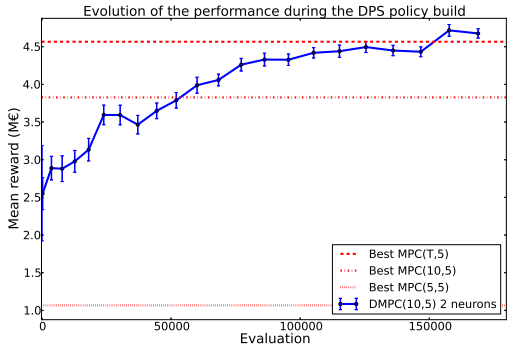


Fig. 1: X-axis: number of evaluations in DPS. Y-axis: reward (the higher the better). Left: Standard volatility. Right: Increased volatility. We see that DPS is far from the performance of MPC, even with the increased volatility; though its computation time per simulation is much shorter (1.5s instead of 18s).

model. The burden of implementing stochastic dynamic programming with its huge computational cost, and, more importantly, the necessary simplified model of random processes, make stochastic dynamic programming tedious and unreliable.**DPS vs MPC: MPC rules.** MPC has strong advantages in terms of simplicity and no need for a learning phase; MPC can directly perform simulations. DPS needs a learning phase; on the other hand, simulations are much faster, and warm starts for reoptimization on alternate test cases are possible. We compare the performance on Fig. 1. *Conclusions:* DPS(5) performs far worse than MPC$(T, 5)$. On the other hand, it is much faster, and MPC(T, o) would be intractable for larger problems (it is not polynomial in o but in the strategic horizon T). DPS(5) performs better than MPC$(5, 5)$, which has the same decision making time. **Mixing DPS and MPC: the best of both worlds.** In the previous section, we have seen that MPC performed much better than DPS, but with an important cost, which would be intractable for larger problems. We work here on the same problem. Fig. 2 present the performance of several variants and combinations of MPC and DPS. *Conclusion:* DMPC(h, o) provides a better performance than MPC(h, o) with the similar decision time. It also performs better than MPC, including the expensive MPC(T, o). DMPC requires an offline learning but MPC(T, o) would become intractable with larger problems.

4 Conclusions

In spite of its simplicity, MPC is a very efficient algorithm with almost no offline learning phase (except the tuning of the quantile parameter q or other parameters of the forecasting modules), but suffers from huge simulation (decision making) time, untractable for large tactical horizon h (e.g. for $h = T$). DPS is usually not (or not easily) compliant with high-dimensional constrained action spaces. We propose a variant (Eq. 5) which is compliant with high-dimensional constrained



With MPC, DPS and DMPC, simulation times essentially depend on h; 1.5s for $h = 5$, 2.6s for $h = 10$, 18s for $h = T$.

Fig. 2: Comparison between MPC, MPC(h) and DMPC(h).

action spaces. **DMPC=DPS+MPC:** When combined with MPC as in Eq. 7, DPS performs quite well (Fig. 2) without increasing the simulation (decision making) time. DMPC(h) is a simple and effective tool for discrete time control, combining the advantages of MPC and DPS (Table 1, experimental results in Fig. 2). DMPC has the best performance overall. **Offline learning phase:** The offline learning phase of DMPC or DPS is the key drawback. This drawback of DMPC or DPS can be mitigated by parallelism. On Fig. 2, only 22 successive iterations are performed; with thousands of cores available, the optimization time boils down to a few minutes. **Further work.** (i) These results are on cases for which MPC with $h = T$ is still tractable. Experiments on bigger testbeds are scheduled. (ii) We will check if DMPC increases its advantage on highly stochastic problems - classical MPC, based on deterministic forecasts, will have difficulties on problems with randomized extreme events.

References

[1] R. Bellman. *Dynamic Programming*. Princeton Univ. Press, 1957.

[2] M. V. F. Pereira and L. M. V. G. Pinto. Multi-stage stochastic optimization applied to energy planning. *Math. Program.*, 52(2):359–375, October 1991.

[3] V. L. de Matos, A. B. Philpott, and E. C. Finardi. Improving the performance of stochastic dual dynamic programming. 2012.

[4] Dimitri P. Bertsekas. Dynamic programming and suboptimal control: A survey from ADP to MPC. *Eur. J. Control*, 11(4-5):310–334, 2005.

[5] M. Zambelli, S. Soares Filho, A.E. Toscano, E. dos Santos, and D. da Silva Filho. NEWAVE versus ODIN: comparison of stochastic and deterministic models for the long term hydropower scheduling of the interconnected brazilian system. *Sba: Sociedade Brasileira de Automatica*, 22:598 – 609, 12 2011.

[6] H.-G. Beyer. *The Theory of Evolutions Strategies*. Springer, Heidelberg, 2001.