Cross-Entropy Optimization of Control Policies with Adaptive Basis Functions

Lucian Buşoniu, Damien Ernst, *Member, IEEE,* Bart De Schutter, *Member, IEEE,* and Robert Babuška

***Abstract*—This paper introduces an algorithm for direct search of control policies in continuous-state, discrete-action Markov decision processes. The algorithm looks for the best closed-loop policy that can be represented using a given number of basis functions (BFs), where a discrete action is assigned to each BF. The type of the BFs and their number are specified in advance and determine the complexity of the representation. Considerable flexibility is achieved by optimizing the locations and shapes of the BFs, together with the action assignments. The optimization is carried out with the cross-entropy method and evaluates the policies by their empirical return from a representative set of initial states. The return for each representative state is estimated using Monte Carlo simulations. The resulting algorithm for cross-entropy policy search with adaptive BFs is extensively evaluated in problems with two to six state variables, for which it reliably obtains good policies with only a small number of BFs. In these experiments, cross-entropy policy search requires vastly fewer BFs than value-function techniques with equidistant BFs, and outperforms policy search with a competing optimization algorithm called DIRECT.**

***Index Terms*—Markov decision processes, direct policy search, adaptive basis functions, cross-entropy optimization.**

I. Introduction

Markov decision processes (MDPs) can be used to model important problems arising in various fields, including automatic control, operations research, computer science, economics, etc. Algorithms to solve general MDPs are therefore very promising for these fields. For instance, in automatic control, such an algorithm would provide a solution to the nonlinear, stochastic optimal control problem [1].

In an MDP, at each discrete time step, the controller measures the state of the process and applies an action according to a control policy.[1] As a result of this action, the process transits into a new state, possibly in a nonlinear or stochastic fashion, and a scalar reward signal is generated that indicates the quality of the transition. The controller measures the new state, and the whole cycle repeats. The control goal is to maximize the cumulative reward (the return) over the course of interaction [1], [2]. Exact solutions can only be found for MDPs with a finite and not too large number of distinct states and actions. In large or continuous-space MDPs, approximate solutions must be sought. In this paper, we focus on such approximate solutions.

The most widely used algorithms for solving MDPs rely on (approximate) value functions, which give the returns from every state or state-action pair [3]. Value functions are often approximated using a linear combination of basis functions (BFs), because this simplifies the theoretical study of the algorithms [4]–[7]. Unfortunately, the value-function based algorithms suffer from significant difficulties in practice. Many algorithms require pre-defined BFs [4], [6], [7], but designing good BFs is a difficult task [5] that requires prior knowledge about the value function. Such prior knowledge is almost never available. Alternatively, many researchers have proposed to automatically change the number, position, or shape of the BFs in order to approximate the value function well, without relying on prior knowledge [5], [8], [9]. However, changing the BFs while estimating the value function can lead to convergence problems.

Motivated by these shortcomings, direct policy search algorithms have been proposed [10]–[13]. These algorithms parameterize the policy and search for an optimal parameter vector that maximizes the return, without using a value function. In the literature, typically ad-hoc policy approximators are designed for specific problems, using intuition and prior knowledge about the optimal policy [10], [12], [14]. The idea of finding good approximators automatically, although extensively explored for value function approximation (as mentioned above), has not often been used in policy search.

In this paper, we develop and evaluate a flexible policy approximator for direct policy search, inspired by the work on automatically finding BFs for value function approximation. The flexibility of this approximator allows it to represent policies for a large class of MDPs. The algorithm works for continuous states and discrete (or discretized) actions. Policies are represented using N state-dependent BFs, where the BFs are associated with discrete actions in a many-to-one fashion. The type of BFs and their number N are specified in advance and determine the complexity of the representation. The locations and shapes of the BFs, together with the action assignments, are the parameters subject to optimization. The optimization criterion is a weighted sum of the returns from a set of representative initial states, where the return for each representative state is computed with Monte Carlo simulations. Each simulated trajectory starts in the representative state considered, and is controlled using the policy to be evaluated; the resulting empirical return is one sample in the Monte Carlo estimate. The representative states together with the weight

Lucian Buşoniu, Bart De Schutter, and Robert Babuška are with the Delft Center for Systems and Control of the Delft University of Technology, The Netherlands (email: {i.l.busoniu, b.deschutter, r.babuska}@tudelft.nl). Bart De Schutter is also with the Marine and Transport Technology Dept. of TU Delft. Damien Ernst is a Research Associate of the Belgian FNRS; he is affiliated with the Systems and Modeling Unit of the University of Liège, Belgium (email: dernst@ulg.ac.be).

The authors wish to thank Louis Wehenkel for his helpful comments. This research was financially supported by the BSIK-ICIS project (grant no. BSIK03024) and by the STW-VIDI project DWV.6188.

[1]Throughout the paper, control-theoretic terms and notations will be preferred to the artificial intelligence terminology often used in the reinforcement learning literature on MDPs. For instance, the terms 'controller' and 'process' will be used instead of 'agent' and 'environment'.

function can be used to focus the algorithm on important parts of the state space.

Mainly due to the rich policy parametrization, the optimization criterion may be a complicated function of the parameter vector, e.g., nondifferentiable and with many local optima. Moreover, both continuous and discrete parameters must be optimized. We select the cross-entropy (CE) method [15] as a powerful optimization tool that is able to solve this difficult problem. The resulting algorithm for CE policy search with adaptive BFs is evaluated in simulation, using three problems: the optimal control of a double integrator, balancing a bicycle riding at a constant speed, and controlling the treatment of infection with the human immunodefficiency virus (HIV). CE policy search is compared with two representative value-function techniques: fuzzy Q-iteration [7] and least-squares policy iteration (LSPI) [6]; and with policy search using a competing optimization algorithm called DIRECT [16].

The remainder of this paper is structured as follows. Section II reviews related work. Section III formally describes MDPs, reviews the algorithms to find MDP solutions, and outlines CE optimization. In Section IV, the CE algorithm for policy search is introduced. Section V reports the results of our simulation experiments. Section VI concludes the paper and provides several directions for future research.

II. Related work

Many policy search approaches focus on gradient-based policy optimization [10], [12], [14], [17]. Actor-critic algorithms also rely on gradient-based policy optimization, but unlike direct policy search they *do* estimate value functions [18]–[21]. Such work is based on the assumption that the locally optimal solution found by the gradient method is good enough, which can be the case when the policy parametrization is simple and well suited for the problem considered. Because of our rich policy parametrization, many local optima may exist and gradient techniques are unsuitable. Instead, we apply the gradient-free CE method for optimization [15], and compare it with DIRECT optimization [16]. Another optimization method that has been applied to policy search is evolutionary computation [22]–[24], Ch. 3 of [13].

Our policy parametrization is inspired by the techniques to automatically find BFs for value function approximation. These techniques include, e.g., BF refinement [5], optimization [25], nonparametric approximators [8], [26], and spectral analysis of the transition function [9]. Out of these options, we choose optimization, but we search for a policy approximator, rather than a value function approximator. In value function approximation, changing the BFs while estimating the value function can lead to a loss of convergence. In our approach, the BFs can be adapted without endangering the convergence of the optimization method to good policy parameters.

CE policy search employs Monte Carlo simulations to evaluate policies, and in this sense it is related to Monte Carlo methods for value function estimation [27]–[29]. Although it converges more slowly than the more popular temporal-difference algorithms (such as Q-learning or SARSA), Monte-Carlo value function estimation is sometimes preferable, e.g., because it is more resilient to incomplete information about the state variables [2].

Using the CE method for policy optimization was first proposed in [11]. In Ch. 4 of [13], a policy was found with the model-reference adaptive search, which is closely related to CE optimization. Both works focus on solving finite, small MDPs, although they also propose solving large MDPs with parameterized policies. In contrast, we focus on solving *continuous-state* MDPs using highly flexible parametrizations based on adaptive BFs. Additionally, we consider representative states associated with weights as a tool to focus the optimization on important initial states, and as a way to circumvent the need to estimate returns for every value of the state, which is impossible when the states are continuous. In [13], only the return starting from one initial state was optimized, whereas in [11] the returns from every (discrete) initial state were optimized.

In [30], we proposed an earlier version of CE policy search with adaptive BFs. Compared to our approach from [30], in the present paper we simplify the policy parametrization, we enhance the algorithm with a smoothing procedure, and we significantly extend the experimental study: the comparisons with LSPI and DIRECT optimization are new, and so are the results for the deterministic bicycle and HIV control.

III. Preliminaries

A. Markov decision processes

In this section, Markov decision processes (MDPs) are formally described, their optimal solution is characterized, and the algorithms to solve MDPs are reviewed [1], [2].

1) MDPs and their solution: An MDP is defined by its state space X, its action space U, its transition probability function $f : X \times U \times X \rightarrow [0,\infty)$, and its reward function $\rho : X \times U \times X \rightarrow \mathbb{R}$. At each discrete time step k, given the state x_k, the controller takes an action u_k according to a control policy $h : X \rightarrow U$. The probability that the next state x_{k+1} belongs to a region $X_{k+1} \subset X$ of the state space is then $\int_{X_{k+1}} f(x_k,u_k,x')\mathrm{d}x'$. For any x and u, $f(x,u,\cdot)$ is assumed to define a valid density[2] of the argument '$\cdot$'. After the transition to x_{k+1}, a reward r_{k+1} is provided according to the reward function ρ: $r_{k+1} = \rho(x_k,u_k,x_{k+1})$. For deterministic MDPs, the transition probability function f is replaced by a simpler transition function, $\overline{f} : X \times U \rightarrow X$, and the reward is completely determined by the current state and action: $r_{k+1} = \overline{\rho}(x_k,u_k)$, $\overline{\rho} : X \times U \rightarrow \mathbb{R}$.

The expected infinite-horizon discounted return for an initial state x_0 under a policy h is:

$$R^h(x_0) = \lim_{K\rightarrow\infty} \mathrm{E}_{x_{k+1}\sim f(x_k,h(x_k),\cdot)} \left\{ \sum_{k=0}^{K} \gamma^k \rho(x_k,h(x_k),x_{k+1}) \right\} \quad (1)$$

where $\gamma \in [0,1)$ is the discount factor. The notation $a \sim p(\cdot)$ means that the random variable a is drawn from the density p. The goal is to find an optimal policy h^* that maximizes the

[2]For simplicity, we will abuse the terminology by using the term 'density' to refer to probability density functions (which describe probabilities of continuous random variables), as well as to probability mass functions (which describe probabilities of discrete random variables).

expected return (1) for every initial state. For any MDP, there exists at least a deterministic optimal policy. Therefore, only *deterministic policies* will be considered in the sequel.

2) Algorithms for solving MDPs: Algorithms that solve MDPs can be organized in two classes: value-function based algorithms and direct policy search algorithms. *Value-function based* algorithms use value functions in order to obtain the optimal policy. For instance, the Q-function (state-action value function) of a policy h gives the expected return when starting in state x, applying action u, and following h thereafter:

$$Q^h(x,u) = \mathrm{E}_{x' \sim f(x,u,\cdot)} \left\{ \rho(x,u,x') + \gamma R^h(x') \right\} \tag{2}$$

The optimal Q-function is defined as $Q^*(x,u) = \max_h Q^h(x,u)$. Any (deterministic) policy that maximizes Q^* in every state x:

$$h^*(x) = \arg\max_u Q^*(x,u) \tag{3}$$

is by definition optimal. A policy that maximizes a Q-function in this way is said to be *greedy* in that Q-function.

The Q-function Q^h satisfies the Bellman equation:

$$Q^h(x,u) = \mathrm{E}_{x' \sim f(x,u,\cdot)} \left\{ \rho(x,u,x') + \gamma Q^h(x', h(u')) \right\} \tag{4}$$

Similarly, the *optimal* Q-function satisfies the Bellman optimality equation:

$$Q^*(x,u) = \mathrm{E}_{x' \sim f(x,u,\cdot)} \left\{ \rho(x,u,x') + \gamma \max_{u'} Q^*(x',u') \right\} \tag{5}$$

Value iteration algorithms solve the Bellman optimality equation (5) iteratively to find the optimal Q-function, and then use (3) to compute an optimal policy. Policy iteration algorithms start with an initial policy. At every iteration they compute the Q-function of the current policy using (4), and then use (3) to find an improved, greedy policy in this Q-function. Some value-function based algorithms use a model of the MDP (the functions f and ρ) [1], [31], while others are model-free and work by only using data [2], [32].

Direct policy search algorithms do not use value functions at all, but represent and optimize the policy directly. Such algorithms are most often used in combination with policy approximation, so we postpone their description until Section III-A3 below, where we discuss solving MDPs approximately.

3) Approximation-based algorithms for large or continuous-space MDPs: The algorithms above require to represent value functions and policies exactly. In general, that is only possible when X and U contain a relatively small number of discrete elements. When X or U are continuous, or discrete but large, *approximations* must be used. Consider first value-function based techniques. In large or continuous state or action spaces, the Q-function cannot be represented exactly, and has to be approximated. Moreover, the Bellman equation involves expectations that must be approximated using only a finite number of samples. Some approximate value-function based algorithms derive from value iteration [4], [5], [7], [8], [33] and others from policy iteration [6], [9]. Usually, the value function approximator is linearly parameterized, because the theoretical study of the algorithms is easier in this case.

Direct policy search algorithms for large or continuous-space MDPs parameterize the policy with a parameter vector a. In principle, policy search algorithms should look for an optimal a^* that maximizes the return (1) for any initial state $x_0 \in X$. When X is large or continuous, computing the return for every initial state is not possible. A practical and commonly used procedure to circumvent this difficulty requires choosing a finite set X_0 of representative initial states. Returns are estimated only for states in X_0, and the optimization criterion is the average return over X_0 [10], [12].

Two approximations must be made to compute the returns for $x \in X_0$. The first approximation replaces the infinite sum in the return (1) with a finite sum over K steps. To ensure that an error of at most $\varepsilon_{\mathrm{MC}}$ (a small positive constant) is introduced, K is chosen with [11]:

$$K = \left\lceil \log_\gamma \frac{\varepsilon_{\mathrm{MC}}(1-\gamma)}{\|\rho\|_\infty} \right\rceil \tag{6}$$

Here, the largest absolute reward $\|\rho\|_\infty$ is assumed finite and $\lceil\cdot\rceil$ gives the smallest integer larger than or equal to the argument (ceiling). The second approximation uses Monte Carlo simulations to estimate the expectation in (1).

B. Cross-entropy optimization

This section briefly introduces the cross-entropy (CE) method for optimization [15]. Consider the following optimization problem:

$$\max_{a \in \mathscr{A}} s(a) \tag{7}$$

where $s : \mathscr{A} \to \mathbb{R}$ is the score function to maximize, and the variable a takes values in the domain $\mathscr{A}$. Let the maximum of s be denoted by s^*. The CE method for optimization maintains a density with support $\mathscr{A}$. In each iteration, a number of samples are drawn from this density and the score values for these samples are computed. A smaller number of samples that have the largest scores are kept, and the remaining samples are discarded. The density is then updated using the selected samples, such that at the next iteration the probability of drawing better samples is increased. The algorithm stops when the score of the worst selected sample no longer improves significantly.

Formally, a family of probability densities $\{p(\cdot; v)\}$ has to be chosen. This family has support $\mathscr{A}$ and is parameterized by v. In each iteration $\tau \geq 1$, a number N_{CE} of samples is drawn from the density $p(\cdot; v_{\tau-1})$, their scores are computed, and the $(1-\rho_{\mathrm{CE}})$ quantile[3] λ_τ of the sample scores is determined, with $\rho_{\mathrm{CE}} \in (0,1)$. Then, a so-called associated stochastic problem is defined, which involves estimating the probability that the score of a sample drawn from $p(\cdot; v_{\tau-1})$ is at least λ_τ:

$$\mathrm{P}_{a \sim p(\cdot; v_{\tau-1})}(s(a) \geq \lambda_\tau) = \mathrm{E}_{a \sim p(\cdot; v_{\tau-1})} \{ I(s(a) \geq \lambda_\tau) \} \tag{8}$$

where I is the indicator function, equal to 1 whenever its argument is true, and 0 otherwise. This is a *stochastic problem* because it requires estimating a probability, in contrast to the deterministic problem (7) to which it is *associated.*

The probability (8) can be estimated by importance sampling. For the associated stochastic problem, an importance

[3]If the score values of the samples are ordered increasingly and indexed such that $s_1 \leq \cdots \leq s_{N_{\mathrm{CE}}}$, then the $(1-\rho_{\mathrm{CE}})$ quantile is: $\lambda_\tau = s_{\lceil (1-\rho_{\mathrm{CE}}) N_{\mathrm{CE}} \rceil}$.

sampling density is one that increases the probability of the interesting event $s(a) \geq \lambda_\tau$. The best importance sampling density in the family $\{p(\cdot;v)\}$, in the sense of the smallest cross-entropy (smallest Kullback-Leibler divergence),[4] is given by the parameter that is the solution of:

$$\arg\max_{v} \mathrm{E}_{a\sim p(\cdot;v_{\tau-1})}\left\{I(s(a)\geq\lambda_\tau)\ln p(a;v)\right\} \tag{9}$$

An approximate, sample-based solution of (9) is found with:

$$v_\tau^\dagger = \arg\max_{v} \frac{1}{N_{\mathrm{CE}}}\sum_{i_{\mathrm{s}}=1}^{N_{\mathrm{CE}}} I(s(a_{i_{\mathrm{s}}})\geq\lambda_\tau)\ln p(a_{i_{\mathrm{s}}};v) \tag{10}$$

This is a Monte-Carlo counterpart of (9), and is therefore called a *stochastic counterpart*.

CE optimization then proceeds with the next iteration using the new density parameter $v_\tau = v_\tau^\dagger$ (note that the probability (8) is never actually computed). The updated density aims to generate good samples with higher probability, thus bringing $\lambda_{\tau+1}$ closer to s^*. The goal is to eventually converge to a density that generates with very high probability samples close to optimal value(s) of a. The algorithm can be stopped when the variation in the $(1-\rho_{\mathrm{CE}})$ quantile does not exceed $\varepsilon_{\mathrm{CE}}$ for d_{CE} successive iterations, or when a maximum number of iterations $\tau_{\max}$ is reached. Here, $\varepsilon_{\mathrm{CE}}$ is a small positive constant and $d_{\mathrm{CE}} > 1, d_{\mathrm{CE}} \in \mathbb{N}$. The largest score among the samples generated in all the iterations is taken as the approximate solution of the optimization problem, and the corresponding sample as an approximate location of an optimum.

Instead of setting the new density parameter equal to the solution $v_\tau^\dagger$ of (10), it can also be updated incrementally:

$$v_\tau = \alpha_{\mathrm{CE}} v_\tau^\dagger + (1-\alpha_{\mathrm{CE}})v_{\tau-1} \tag{11}$$

where $\alpha_{\mathrm{CE}} \in (0,1]$. This so-called smoothing procedure is useful to prevent CE optimization from becoming stuck in local optima [15].

Under certain assumptions on $\mathscr{A}$ and $p(\cdot;v)$, the stochastic counterpart (10) can be solved analytically. One particularly important case when this happens is when $p(\cdot;v)$ belongs to the natural exponential family. For instance, when $\{p(\cdot;v)\}$ is the family of Gaussians parameterized by the mean η and the standard deviation σ (so, $v = [\eta,\sigma]^{\mathrm{T}}$), the solution v_τ of (10) consists of the mean and the standard deviation of the best samples, i.e., of the samples $a_{i_{\mathrm{s}}}$ for which $s(a_{i_{\mathrm{s}}}) \geq \lambda_\tau$.

While the convergence of CE optimization is not guaranteed in general, the algorithm is usually convergent in practice [15]. For combinatorial (discrete-variable) optimization, the CE method provably converges with probability 1 to a unit mass density, which always generates samples equal to a single point. Furthermore, the probability that this convergence point is in fact an optimal solution can be made arbitrarily close to 1 by using a sufficiently small smoothing parameter α_{CE} [34].

[4]In general, the Kullback-Leibler divergence between two densities q and p is $\mathrm{E}_{a\sim q}\{\ln[q(a)/p(a)]\} = \mathrm{E}_{a\sim q}\{\ln q(a)\} - \mathrm{E}_{a\sim q}\{\ln p(a)\}$. For an explanation on how the Kullback-Leibler divergence is used to arrive at (9), see [15].

IV. CE POLICY SEARCH WITH ADAPTIVE BASIS FUNCTIONS

This section describes the proposed algorithm for policy optimization using the CE method. First, the policy parametrization and the performance index (score function) are discussed. Then, a general algorithm is given, followed by an instantiation that uses Gaussian radial basis functions (RBFs) (a type of BFs described by (15) below) to parameterize policies.

A. Policy parametrization and score function

Consider a stochastic or deterministic MDP. Denote by D the number of state variables of the MDP (i.e., the dimension of X). In the sequel, it is assumed that the action space of the MDP is discrete and contains M distinct actions, $U_{\mathrm{d}} = \{u_1,\ldots,u_M\}$. The set U_{d} can result from the discretization[5] of an originally larger (e.g., continuous) action space U.

The policy parametrization uses N basis functions (BFs) defined over the state space. The BFs are parameterized by a vector $\xi \in \Xi$ that typically gives their locations and shapes. Denote these BFs by $\varphi_i(x;\xi): X \to \mathbb{R}$, $i = 1,\ldots,N$, to highlight their dependence on ξ. The BFs are associated to discrete actions by a many-to-one mapping. This mapping can be represented as a vector $\vartheta \in \{1,\ldots,M\}^N$ that associates each BF φ_i to a discrete action index ϑ_i, or equivalently to a discrete action u_{ϑ_i}. A schematic representation of this parametrization is given in Figure 1. The policy chooses for any x the action associated to the BF that takes the largest value at x:

$$h(x) = u_{\vartheta_{i^*}}, \quad i^* = \arg\max_{i} \varphi_i(x;\xi) \tag{12}$$

Here, i^* is the index of the largest BF at x, while ϑ_{i^*} is the index of the discrete action associated with this BF. Any ties in the maximization should be broken consistently, e.g., always in favor of the smallest BF index.

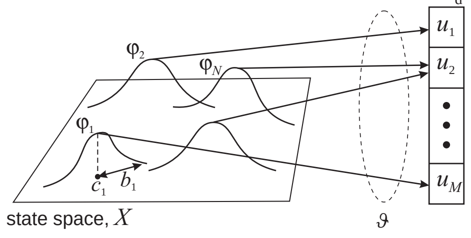

Fig. 1. A schematic representation of the policy parametrization. The vector ϑ associates the BFs to the discrete actions. In this example, the BFs are parameterized by their centers c_i and widths b_i, so that $\xi = [c_1^{\mathrm{T}}, b_1^{\mathrm{T}}, \ldots, c_N^{\mathrm{T}}, b_N^{\mathrm{T}}]^{\mathrm{T}}$.

This policy is parameterized by the vector $[\xi^{\mathrm{T}}, \vartheta^{\mathrm{T}}]^{\mathrm{T}}$, ranging in the set $\Xi \times \{1,\ldots,M\}^N$. CE optimization is used to search for an optimal parameter vector $[\xi^{*\mathrm{T}}, \vartheta^{*\mathrm{T}}]^{\mathrm{T}}$ that maximizes the following score function:

$$s(\xi,\vartheta) = \sum_{x_0\in X_0} w(x_0)\widehat{R}^h(x_0) \tag{13}$$

[5]Discretization is a standard technique for solving continuous-action MDPs. In this paper, we assume that an appropriate discretization is available, and focus on optimizing the policy *given* this discretization. In general, care should be taken when discretizing the actions, as doing so incorrectly can reduce the performance or even prevent finding a good policy.

where $\widehat{R}^h$ is the estimated return of the policy h determined by ξ and ϑ, and X_0 is a given finite set of representative states, weighted by $w : X_0 \to (0, 1]$ (see also Section III-A3).[6] The return for each state in X_0 is estimated by Monte Carlo simulations:

$$\widehat{R}^h(x_0) = \frac{1}{N_{\text{MC}}} \sum_{i_0=1}^{N_{\text{MC}}} \sum_{k=0}^{K} \gamma^k \rho(x_{i_0,k}, h(x_{i_0,k}), x_{i_0,k+1}) \tag{14}$$

where $x_{i_0,0} = x_0$, $x_{i_0,k+1} \sim f(x_{i_0,k}, h(x_{i_0,k}), \cdot)$, and N_{MC} is the number of Monte Carlo simulations to carry out. So, each simulation i_0 makes use of a system trajectory that is K steps long and generated using the policy h. The system trajectories are generated independently. The length K is chosen with (6) to guarantee that an error of at most ε_{MC} is introduced by truncating the trajectory. The parameter $\varepsilon_{\text{MC}} > 0$ can be chosen a few orders of magnitude smaller than the typical return obtained from the states in X_0. For deterministic MDPs, a single trajectory is simulated for every initial state in X_0, so $N_{\text{MC}} = 1$. For stochastic MDPs, several trajectories should be simulated, $N_{\text{MC}} > 1$, with a good value of N_{MC} depending on the MDP considered.

The performance of the resulting policy is determined by two essential choices made by the user: the number of BFs, together with their type; and the set of representative states, together with their weight function. Next, we briefly discuss these two choices.

The number N of BFs, in combination with the chosen type of BFs, determine the accuracy of the policy approximator. Given the type of BFs, in general a good value of N for a given problem cannot be determined in advance, but must be found empirically. Fortunately, as will become apparent in Section V, at least for the example problems that we study, a relatively small number of BFs is sufficient to provide a good policy approximation. In two of the examples, we also study the effect of varying N. Note that, in special cases, prior knowledge about the complexity of an optimal policy may be available, which could be exploited to choose beforehand a reasonable type of BFs and value of N (e.g., in the academic, double-integrator problem of Section V-A, an accurate representation of an optimal policy can be found by exhaustive search).

The second choice that must be made concerns the set X_0 and the weight function w. The set X_0 should consist of (a representative selection of) the initial states from which the system must be controlled near-optimally. For instance, some problems only require to control the system well from a restricted set of initial states; X_0 should then be equal to this set, or included in it when the set is too large. Initial states that are deemed more important can be assigned larger weights. When all initial states are equally important, the elements of X_0 should be uniformly spread over the state space, e.g., equidistantly, and identical weights equal to $\frac{1}{|X_0|}$ should be assigned to every element of X_0 ($|\cdot|$ denotes set cardinality). Since not all of the interesting initial states can always be included in X_0, an important question is how the resulting policy performs when applied from initial states that *do not* necessarily belong to X_0, i.e., how it *generalizes* to unseen initial states. We study the influence of X_0, as well as policy generalization, for the bicycle problem in Section V-B.

[6]More generally, a density w over the initial states can be considered, and the score function is then $\mathrm{E}_{x_0 \sim w(\cdot)} \left\{ R^h(x_0) \right\}$, i.e., the expected return when $x_0 \sim w(\cdot)$. Such a score function can be estimated by Monte Carlo methods. In this paper, we only use finite sets X_0 associated with weighting functions w as in (13).

Note that the policy representation can associate multiple BFs to a single action, and also allows actions that are not assigned to any BF. The former mechanism is needed to represent the (potentially complicated) regions of the state space where an action is optimal. The latter mechanism can remove from the policy representation any discrete actions that are not needed to near-optimally control the system.

B. The general algorithm for CE policy search

In order to apply the CE method to the problem of finding optimal parameters ξ^* and ϑ^* that maximize (13), an associated stochastic problem (8) must be defined. To this end, it is necessary to choose a family of densities with support $\Xi \times \{1, \ldots, M\}^N$. This paper focuses on continuous-state MDPs, for which the BFs typically have continuous parameters. This means that Ξ is a continuous set, whereas $\{1, \ldots, M\}^N$ is discrete. Rather than using a density with mixed (partly continuous and partly discrete) support, it is convenient to employ separate densities for the two parts ξ and ϑ of the parameter vector: a density $p_\xi(\cdot; v_\xi)$ for ξ, parameterized by v_ξ and with continuous support Ξ, and a density $p_\vartheta(\cdot; v_\vartheta)$ for ϑ, parameterized by v_ϑ and with discrete support $\{1, \ldots, M\}^N$. Let N_{v_ξ} be the number of elements in the vector v_ξ, and N_{v_ϑ} the number of elements in v_ϑ.

Algorithm 1 Cross-entropy policy search.

```
Input: transition & reward functions $f$, $\rho$, discount factor $\gamma$,
    representative states $X_0$, weight function $w$,
    density families $\{p_\xi(\cdot; v_\xi)\}, \{p_\vartheta(\cdot; v_\vartheta)\}$,
    number of density parameters $N_{v_\xi}$, $N_{v_\vartheta}$,
    initial density parameters $v_{\xi,0}$, $v_{\vartheta,0}$,
    parameters $N$, $\rho_{\text{CE}}$, $c_{\text{CE}}$, $\alpha_{\text{CE}}$, $d_{\text{CE}}$, $\varepsilon_{\text{CE}}$, $\varepsilon_{\text{MC}}$, $N_{\text{MC}}$, $\tau_{\max}$
1: $N_{\text{CE}} \leftarrow c_{\text{CE}}(N_{v_\xi} + N_{v_\vartheta})$; $\tau \leftarrow 0$
2: repeat
3:     $\tau \leftarrow \tau + 1$
4:     Generate samples $\xi_1, \ldots, \xi_{N_{\text{CE}}}$ from $p_\xi(\cdot; v_{\xi,\tau-1})$
5:     Generate samples $\vartheta_1, \ldots, \vartheta_{N_{\text{CE}}}$ from $p_\vartheta(\cdot; v_{\vartheta,\tau-1})$
6:     Compute $s(\xi_{i_s}, \vartheta_{i_s})$ with (13), $i_s = 1, \ldots, N_{\text{CE}}$
7:     Reorder and reindex s.t. $s_1 \le \cdots \le s_{N_{\text{CE}}}$
8:     $\lambda_\tau \leftarrow s_{\lceil (1-\rho_{\text{CE}}) N_{\text{CE}} \rceil}$
9:     $v^\dagger_{\xi,\tau} \leftarrow \arg\max_{v_\xi} \sum_{i_s=\lceil (1-\rho_{\text{CE}}) N_{\text{CE}} \rceil}^{N_{\text{CE}}} \ln p_\xi(\xi_{i_s}; v_\xi)$
10:    $v^\dagger_{\vartheta,\tau} \leftarrow \arg\max_{v_\vartheta} \sum_{i_s=\lceil (1-\rho_{\text{CE}}) N_{\text{CE}} \rceil}^{N_{\text{CE}}} \ln p_\vartheta(\vartheta_{i_s}; v_\vartheta)$
11:    $v_{\xi,\tau} \leftarrow \alpha_{\text{CE}} v^\dagger_{\xi,\tau} + (1 - \alpha_{\text{CE}}) v_{\xi,\tau-1}$
12:    $v_{\vartheta,\tau} \leftarrow \alpha_{\text{CE}} v^\dagger_{\vartheta,\tau} + (1 - \alpha_{\text{CE}}) v_{\vartheta,\tau-1}$
13: until ($\tau > d_{\text{CE}}$ and $|\lambda_{\tau-\tau'} - \lambda_{\tau-\tau'-1}| \le \varepsilon_{\text{CE}}$, for $\tau' =$
        $0, \ldots, d_{\text{CE}} - 1$) or $\tau = \tau_{\max}$
Output: $\widehat{\xi}^*, \widehat{\vartheta}^*$, the best sample; and $\widehat{s}^* = s(\widehat{\xi}^*, \widehat{\vartheta}^*)$
```

The CE algorithm for policy search is given as Algorithm 1. For easy reference, Table I collects the meaning of the parameters and variables of CE policy search. The score

TABLE I
PARAMETERS AND VARIABLES FOR CE POLICY SEARCH

Symbol	Meaning
N; M	number of BFs; number of discrete actions
ξ; ϑ	BF parameters; assignment of discrete actions to BFs
v_ξ; v_ϑ	parameters of the density for ξ; and for ϑ
N_{CE}	number of samples used at every CE iteration
ρ_{CE}	proportion of samples used in the CE updates
λ	$(1-\rho_{\mathrm{CE}})$ quantile of the sample performance
c_{CE}	multiple of the number of density parameters
α_{CE}	smoothing parameter
d_{CE}	how many iterations λ should remain roughly constant
$\varepsilon_{\mathrm{CE}}$; $\varepsilon_{\mathrm{MC}}$	convergence threshold; precision in computing returns
N_{MC}	number of Monte Carlo simulations for each state
τ; $\tau_{\max}$	iteration index; maximum number of iterations

computation at line 6 of Algorithm 1 involves the Monte Carlo estimation (14) of the return for every representative initial state, using the dynamics f to generate the necessary trajectories. The stochastic counterparts in lines 9 and 10 were simplified, using the fact that the samples are already sorted in the ascending order of their scores. The algorithm terminates when the variation of λ is at most $\varepsilon_{\mathrm{CE}}$ for d_{CE} consecutive iterations, or when a maximum number $\tau_{\max}$ of iterations has been reached. The integer $d_{\mathrm{CE}} > 1$ ensures that the performance variation does not decrease below $\varepsilon_{\mathrm{CE}}$ accidentally, but that it remains steady for d_{CE} iterations.

Many times it is convenient to use densities with unbounded support (e.g., Gaussians) when the BF parameters are continuous. However, the set Ξ must typically be bounded, e.g., when ξ contains centers of RBFs, which must remain inside a bounded state space. Whenever this situation arises, samples can be generated from the density with larger support, and those samples that do not belong to Ξ can be rejected. The procedure continues until N_{CE} valid samples are generated, and the rest of the algorithm remains unchanged. The situation is entirely similar for the discrete action assignments ϑ, when it is convenient to use a family of densities $p_\vartheta(\cdot; v_\vartheta)$ with a support larger than $\{1, \ldots, M\}^N$. An equivalent algorithm that uses all the samples can always be given by extending the score function to give samples falling outside the domain very large negative scores (larger in magnitude than for any valid sample). Additionally, for this equivalent algorithm, N_{CE} and ρ_{CE} have to be adapted at each iteration such that a constant number of valid samples is generated, and that a constant number of best samples is used for the parameter updates. The theoretical basis of CE optimization is therefore not affected by sample rejection.

The most important parameters in CE policy search are, like in CE optimization, the number of samples N_{CE} and the proportion of best samples used to update the density, ρ_{CE}. The parameter c_{CE} is taken greater than or equal to 2, so that the number of samples is a multiple of the number of density parameters [15]. The parameter ρ_{CE} can be taken around 0.01 for large numbers of samples, or larger, around $(\ln N_{\mathrm{CE}})/N_{\mathrm{CE}}$, if there are only a few samples ($N_{\mathrm{CE}} < 100$) [15]. Since it does not make sense to impose a convergence threshold smaller than the precision of the score function, $\varepsilon_{\mathrm{CE}}$ should be chosen larger than or equal to $\varepsilon_{\mathrm{MC}}$. A good value is $\varepsilon_{\mathrm{CE}} = \varepsilon_{\mathrm{MC}}$.

C. CE policy search with radial basis functions

In this section, we describe an instantiation of CE policy search that uses state-dependent Gaussian RBFs and a binary representation of the action assignments. We choose Gaussian RBFs because they are commonly used to represent approximate MDP solutions [4], [6], [25], [33]. Many other types of BFs could be used instead, including, e.g., splines and polynomials. We assume that the state space is a hyperbox centered in the origin: $X = \left\{x \in \mathbb{R}^D \,\middle|\, |x| \le x_{\max}\right\}$, where $x_{\max} \in (0, \infty)^D$, and where the absolute value and relational operators are applied element-wise. This assumption is made here for simplicity and can easily be relaxed.[7]

A special case arises when the problem has terminal states, i.e., states that terminate process trajectories. Formally, applying any action from a terminal state returns the process to the same state with zero reward, so that the remaining subtrajectory is irrelevant in the return (1), see Sec. 3.4 of [2]. In this case, we assume that the set of *nonterminal* states is a hyperbox, and by a slight abuse of notation denote this nonterminal state space by X. Since action choices need not be made in terminal states, ignoring them in the policy parametrization does not limit the algorithm.

The Gaussian RBFs are defined by:

$$\varphi_i(x; \xi) = \exp\left[-\sum_{d=1}^{D} \frac{(x_d - c_{i,d})^2}{b_{i,d}^2}\right] \tag{15}$$

where D is the number of state variables, $c_i = [c_{i,1}, \ldots, c_{i,D}]^{\mathrm{T}}$ is the D-dimensional center, and $b_i = [b_{i,1}, \ldots, b_{i,D}]^{\mathrm{T}}$ the D-dimensional radius of the i-th RBF. Denote the vector of centers by $c = [c_1^{\mathrm{T}}, \ldots, c_N^{\mathrm{T}}]^{\mathrm{T}}$ and the vector of radii by $b = [b_1^{\mathrm{T}}, \ldots, b_N^{\mathrm{T}}]^{\mathrm{T}}$. So, $c_{i,d}$ and $b_{i,d}$ are scalars, c_i and b_i are D-dimensional vectors that collect the scalars for all D dimensions, and c and b are DN-dimensional vectors that collect the D-dimensional vectors for all N RBFs. The centers of the RBFs must lie within the bounded state space X, and the radii must be strictly positive, i.e., $c \in X^N$ and $b \in (0, \infty)^{DN}$. The BFs parameter vector is therefore $\xi = [c^{\mathrm{T}}, b^{\mathrm{T}}]^{\mathrm{T}}$ and takes values in $\Xi = X^N \times (0, \infty)^{DN}$.

To define the associate stochastic problem for CE optimization, independent univariate Gaussian densities are selected for each element of the parameter vector ξ. Gaussian densities are commonly used in continuous-variable CE optimization, see, e.g., Ch. 5 of [15]. They allow the CE procedure to converge to a precise optimum location, by letting the standard deviation of each univariate density converge to zero. The density for each center $c_{i,d}$ is parameterized by the mean $\eta_{i,d}^c$ and standard deviation $\sigma_{i,d}^c$; and the density for each radius $b_{i,d}$ is parameterized by the mean $\eta_{i,d}^b$ and the standard deviation $\sigma_{i,d}^b$. Similarly to the centers and radii themselves, we denote the DN-dimensional vectors of means and standard deviations respectively by η^c, σ^c for the RBF centers, and by η^b, σ^b for the RBF radii. The parameter for the density $p_\xi(\cdot; v_\xi)$ is then $v_\xi = [(\eta^c)^{\mathrm{T}}, (\sigma^c)^{\mathrm{T}}, (\eta^b)^{\mathrm{T}}, (\sigma^b)^{\mathrm{T}}]^{\mathrm{T}} \in \mathbb{R}^{4DN}$. Since the support of this density is $\mathbb{R}^{2DN}$ instead of the

[7]It will be relaxed, e.g., when CE policy search is applied to the HIV infection control problem of Section V-C.

required $\Xi = X^N \times (0,\infty)^{DN}$, samples that do not belong to Ξ are rejected and generated again.

The means and standard deviations for the RBF centers and radii are initialized for all i as follows:

$$\eta_i^c = 0_D, \quad \sigma_i^c = x_{\max}, \quad \eta_i^b = \frac{1}{2(N+1)} \cdot x_{\max}, \quad \sigma_i^b = \eta_i^b$$

where 0_D is a vector of D zeros. The density parameters for the RBF centers are initialized to ensure a good coverage of the state space, while the parameters for the RBF radii are initialized heuristically to have a similar overlap between RBFs as N varies. At the end of each iteration τ of Algorithm 1, the means and standard deviations are updated element-wise, as the means and standard deviations of the best samples (see Section III-B).

The vector ϑ, which contains the assignments of discrete actions to BFs, is represented in binary code. Each element ϑ_i is represented using $N^{\text{bin}} = \lceil \log_2 M \rceil$ bits, so that the binary representation of ϑ has NN^{bin} bits. Every bit is drawn from a Bernoulli distribution parameterized by its mean $\eta^{\text{bin}} \in [0,1]$ (η^{bin} gives the probability of selecting 1; the probability of selecting 0 is $1-\eta^{\text{bin}}$). Similarly to the Gaussian densities above, such a concatenation of Bernoulli distributions can converge to a degenerate distribution that always generates samples equal to a precise optimum. Note that, if M is not a power of 2, bit combinations corresponding to invalid indices are rejected and generated again. Because every bit has its own Bernoulli parameter, the total number of Bernoulli parameters v_ϑ is NN^{bin}. The Bernoulli distribution belongs to the natural exponential family, so the updated density parameters $v_{\vartheta,\tau}$ in line 10 of Algorithm 1 can be computed analytically, as the mean of the best samples in their binary representation.

Now, we briefly examine the complexity of CE policy search with RBFs. The number of density parameters is $N_{v_\xi} = 4DN$ for the RBF centers and radii, and $N_{v_\vartheta} = NN^{\text{bin}}$ for the action assignments. Therefore, the total number of samples used is $N_{\text{CE}} = c_{\text{CE}}(4DN + NN^{\text{bin}})$. The largest amount of computation is spent by the algorithm in the simulations used to estimate the score of each sample. Neglecting therefore the other computations, the complexity of one CE iteration is at most:

$$t_{\text{step}}[c_{\text{CE}}N(4D+N^{\text{bin}}) \cdot |X_0| \cdot N_{\text{MC}}K] \tag{16}$$

where K is the maximum length of each trajectory, and t_{step} is the time needed to compute $h(x)$ for a fixed x and to simulate the controlled system for one time step.

V. Experimental studies

In the sequel, the performance of CE policy search is assessed using extensive numerical experiments, for three problems that gradually increase in dimensionality: optimal control of a double integrator (two dimensions, Section V-A), balancing a bicycle that rides at a constant speed (four dimensions, Section V-B), and controlling the treatment of infection with the human immunodefficiency virus (HIV) (six dimensions, Section V-C). Using the lower-dimensional problems, we investigate the influence of the number of BFs and of the choice of representative states, and we compare CE policy search with alternative algorithms. HIV infection control illustrates that CE policy search also works in a realistic, highly challenging problem.

A. Discrete-time double integrator

In this section, a double integrator optimal control problem is used to evaluate CE policy search. This problem is stated such that (near-)optimal trajectories from any state terminate in a small number of steps. This property allows extensive simulation experiments to be run and an optimal solution to be found without excessive computational costs.

1) Model and an optimal policy: The double integrator has the (nonterminal) state space $X = [-1,1] \times [-0.5,0.5]$, a binary action space $U_{\text{d}} = \{-0.1, 0.1\}$, and the deterministic dynamics:

$$x_{k+1} = \overline{f}(x_k,u_k) = \begin{bmatrix} x_{1,k} + x_{2,k} \\ \min\big(\max(x_{2,k}+u_k,-0.5),0.5\big) \end{bmatrix} \tag{17}$$

where $x_{d,k}$ denotes the dth state variable at time k, and the operators 'min' and 'max' are applied to restrict the evolution of the velocity x_2 to $[-0.5,0.5]$. The states for which $|x_1| > 1$ are terminal. The goal is to drive the position x_1 beyond either boundary of the interval $[-1,1]$ (i.e., to a terminal state), so that when x_1 crosses the boundary, x_2 is as small as possible in magnitude. This goal is expressed by the reward function:

$$r_{k+1} = \overline{\rho}(x_k,u_k) = -(1-\tilde{x}_{1,k+1})^2 - x_{2,k+1}^2 \tilde{x}_{1,k+1}^2 \tag{18}$$

where $\tilde{x}_{1,k+1} = \min(|x_{1,k+1}|, 1)$ so that terminal states are equally rewarded regardless of how far beyond the interval $[-1,1]$ they are. The discount factor γ is set to 0.95.

Figure 2 gives an accurate representation of an optimal policy for this problem, consisting of the optimal actions for a regular grid of 101×101 points covering the (nonterminal) state space. The optimal actions were obtained using the following brute-force procedure, made possible because the problem has terminal states, and the optimal trajectories from any initial state terminate in a small number of steps. All the possible sequences of actions of a sufficient length were generated, and the system was controlled with all these sequences starting from every state on the grid. The length is sufficient if it is larger than the lengths of optimal trajectories from any initial state, and is determined empirically. For every state on the grid, a sequence that produced the best discounted return is by definition optimal, and the first action in this sequence is an optimal action.

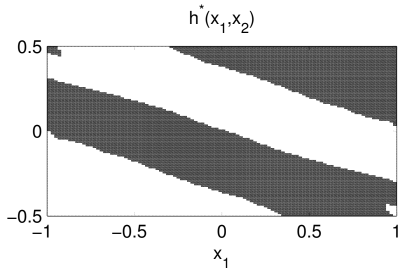

Fig. 2. An optimal policy for the double integrator. Black corresponds to the action -0.1, and white to $+0.1$.

The policy in Figure 2 should be representable with a moderate number of RBFs. In particular, if the small-scale variations in the top-left and bottom-right corners of the graph are disregarded, the policy consists of only four constant-action "stripes". Since the RBFs are axis-aligned while the stripes are diagonally-oriented, one RBF is insufficient to approximate each stripe; a few RBFs should, however, suffice.

2) Results of CE policy search: To apply CE policy search, the following set of representative initial states was used to compute the score (13):

$$X_0 = \{-1, -0.9, \dots, 1\} \times \{-0.5, -0.3, -0.1, 0, 0.1, 0.3, 0.5\}$$

This set contains $21 \times 7 = 147$ states, fewer than the grid of Figure 2. The states were equally weighted using $w(x_0) = 1/|X_0|$ for any x_0. The parameter settings for the algorithm were $c_{\text{CE}} = 10$, $\rho_{\text{CE}} = 0.01$, $\alpha_{\text{CE}} = 0.7$, $\varepsilon_{\text{CE}} = \varepsilon_{\text{MC}} = 0.001$, $d_{\text{CE}} = 5$, $\tau_{\max} = 100$. Little or no tuning was necessary to choose these values. Because the system is deterministic, $N_{\text{MC}} = 1$. With these parameter settings, CE policy search was run while gradually increasing the number N of BFs from 4 to 18. For every value of N, 20 independent runs were performed. The algorithm always converged before reaching the maximum number of iterations.

Figure 3 presents the performance of the policies obtained by CE policy search. The mean values across the 20 runs are shown, together with their 95% confidence intervals. For comparison, the figure also shows the exact optimal score for X_0, computed by looking for optimal open-loop action sequences with the procedure explained in Section V-A1. Figure 4 shows the execution time of the algorithm.[8]

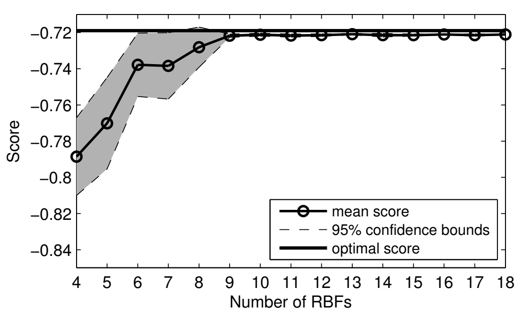

Fig. 3. Performance of CE policy search.

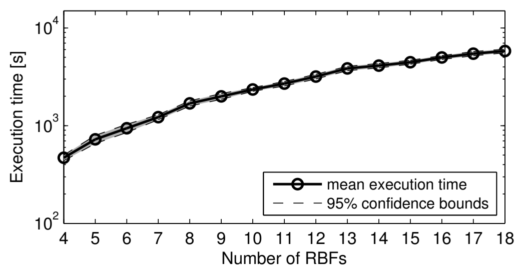

Fig. 4. Execution time of CE policy search.

CE policy search reliably obtains near-optimal performance for $N \geq 9$. This shows that, as intuition indicated, the optimal policy can indeed be represented well using only a small number of (optimized) RBFs. The execution time is roughly affine in N, as expected from (16).

[8]All the computation times reported in this paper were recorded while running the algorithms in MATLAB 7 on a PC with an Intel Core 2 Duo E6550 2.33 GHz CPU and with 3 GB RAM.

3) Comparison with value-function based algorithms: In this section, CE policy search is compared with fuzzy Q-iteration [7], which is a representative value iteration algorithm, and with least-squares policy iteration (LSPI) [6], which is a representative policy iteration algorithm (see Section III-A). *Fuzzy Q-iteration* relies on a linearly-parameterized Q-function approximator with N state-dependent BFs $\phi_1, \dots, \phi_N : X \to \mathbb{R}$, which are replicated for each discrete action $u_j \in U_{\text{d}}$. Approximate Q-values are computed with:

$$\widehat{Q}(x, u_j) = \sum_{i=1}^{N} \phi_i(x) \theta_{i,j} \tag{19}$$

where $\theta \in \mathbb{R}^{N \times M}$ is a matrix of parameters. Fuzzy Q-iteration computes an approximately optimal Q-function using an approximation of the Bellman optimality equation (5), and then outputs the greedy policy (3) in this Q-function. It converges to a solution with a bounded suboptimality. For the double integrator example, we defined triangular BFs distributed on an equidistant grid with N' points along each dimension of X; this led to a total number of $N = (N')^2$ state-dependent BFs, and $2(N')^2$ BFs for both discrete actions. In the lack of prior knowledge, such a regular placement of BFs is a good choice, because it provides a uniform resolution over the state space.

From the class of policy iteration algorithms, *LSPI* is selected [6]. This algorithm uses (19) to approximate the Q-function of the current policy (rather than the optimal Q-function, as fuzzy Q-iteration did). To find the Q-function parameters θ, LSPI solves a *projected* version of the Bellman equation (4). The coefficients of this equation are estimated from transition samples. Once an approximate Q-function is available, LSPI improves the policy using (3). Then, it estimates the Q-function of the improved policy, and so on. The sequence of policies produced by LSPI eventually converges to a subsequence along which all the policies have a bounded suboptimality [6]. For the double integrator example, we defined normalized Gaussian RBFs. Like for fuzzy Q-iteration above, the centers of the RBFs were placed on an equidistant grid with N' points along each dimension of X. The radii of the RBFs along each dimension were taken identical to the grid spacing along that dimension. A total number of $2(N')^2$ state-action BFs was obtained.

For both fuzzy Q-iteration and LSPI, the number N' of BFs for each state variable was gradually increased from 4 to 18. Fuzzy Q-iteration is a deterministic algorithm, hence it was run once for every N'. LSPI requires a set of random samples, so each LSPI experiment was run 20 times with independent sets of samples. For $N' = 4$, 1000 samples were used, and for larger N' the number of samples was increased proportionally with the number $2(N')^2$ of parameters: thus, $\left\lceil \frac{(N')^2}{4^2} \cdot 1000 \right\rceil$ samples were used for each N'. The performance of the policies computed by fuzzy Q-iteration and LSPI is shown in Figure 5 (compare with Figure 3), and the execution time of the algorithms in Figure 6 (compare with Figure 4).

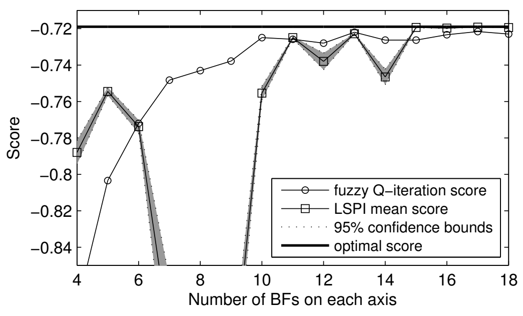

Fig. 5. Performance of fuzzy Q-iteration and LSPI (performances below −0.85 are not shown).

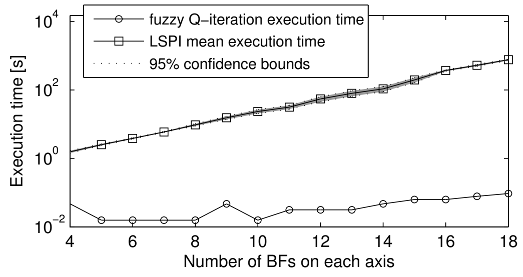

Fig. 6. Execution time of fuzzy Q-iteration and LSPI.

Whereas CE policy search reliably obtained near-optimal performance starting from $N = 9$ BFs *in total*, fuzzy Q-iteration and LSPI obtain good performance starting from around $N' = 11$ BFs *on each axis*; the total number of BFs is $2(N')^2$ (much larger). This difference is mainly due to the fact that the BFs used by fuzzy Q-iteration and LSPI are equidistant and identically shaped, whereas the CE algorithm optimizes the shapes and locations of the BFs.

The computational cost of the value-function algorithms is smaller (for fuzzy Q-iteration, by several orders of magnitude) than the cost of CE policy search. This indicates that, in low-dimensional problems such as the double integrator, CE policy search should preferably be used when a flexible policy approximator having a fixed complexity has to be found, and the computational costs to optimize this fixed-complexity approximator are not a concern.

4) Comparison of CE and DIRECT optimization: In our policy search approach, a global, mixed-integer, gradient-free optimization problem must be solved. One algorithm that can address this difficult optimization problem is DIRECT [16]. In this section, we compare CE optimization with DIRECT in the context of policy search. DIRECT works in hyperbox parameter spaces such as those considered in this paper, by recursively splitting promising hyperboxes in three and sampling the center of each resulting hyperbox. The hyperbox selection procedure leads both to a global exploration of the parameter space, and to a local search in the most promising regions discovered so far. The algorithm is especially suitable for problems in which evaluating the score function is computationally costly [16], as is the case in policy search.[9]

[9]We use the DIRECT implementation from the TOMLAB optimization toolbox for MATLAB.

We used DIRECT to optimize the parameters of the policy (12) while increasing N from 4 to 18, like for CE optimization above. DIRECT stops when the score function (13) has been evaluated a given number of times; this stopping parameter was set to $2000 \cdot 5N$ for every N, i.e., 2000 times the number of parameters to optimize. Since DIRECT is a deterministic algorithm, each experiment was run only once. The performance of the policies computed by DIRECT is shown in Figure 7, and the execution time of the algorithm in Figure 8. For an easy comparison, the CE policy search results from Figures 3 and 4 are also repeated.

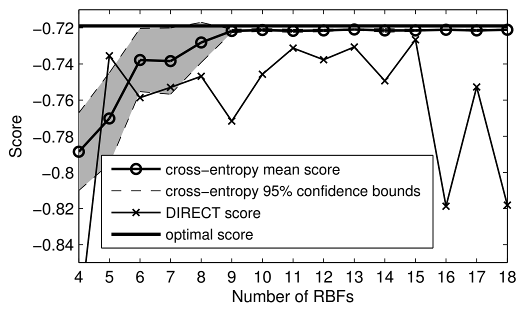

Fig. 7. Performance of DIRECT — comparison with CE optimization.

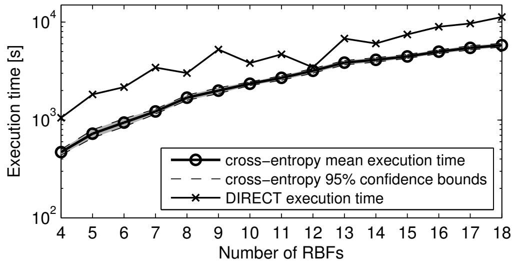

Fig. 8. Execution time of DIRECT — comparison with CE optimization.

DIRECT performs worse than CE optimization for most values of N, while requiring more computations for all values of N. Also, unlike CE optimization, DIRECT does not reliably improve the performance as N increases. Increasing the allowed number of score evaluations may possibly improve the performance of DIRECT, but would also make it even more computationally expensive, and therefore less competitive with CE optimization. The poor results of DIRECT, including the high variability of its performance with increasing N, may be due to its reliance on splitting the parameter space into hyperboxes: this approach can perform poorly when the parameter space is high-dimensional, as is the case for policy search.

B. Bicycle balancing

Next, CE policy search is applied to a more involved problem than the double integrator: balancing a bicycle that rides at a constant speed on a horizontal surface (Figure 9). This is a variant of the bicycle balancing and riding problem, a popular benchmark for algorithms that solve MDPs [6], [8], [35]. The steering column of the bicycle is vertical, which means the bicycle is not self-stabilizing, but must be actively stabilized to prevent it from falling. The state vector is $[\omega, \dot{\omega}, \alpha, \dot{\alpha}]^{\mathrm{T}}$, where ω [rad] is the roll angle of the bicycle measured from the vertical axis, α [rad] is the angle of the handlebar, equal to 0 when the handlebar is in its neutral position, and $\dot{\omega}$, $\dot{\alpha}$ [rad/s] are the respective angular velocities. The control variables are the displacement $\delta \in [-0.02, 0.02]$ m of the bicycle-rider common center of mass perpendicular to the plane of the bicycle, and the torque $\tau \in [-2, 2]$ Nm applied to the handlebar. The displacement δ can be affected by additive noise ν drawn from a uniform density over the interval $[-0.02, 0.02]$ m. For more details about the bicycle problem and its (nonlinear) dynamical model, see [8], [35].[10]

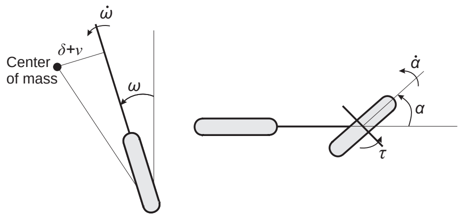

Fig. 9. A schematic representation of the bicycle seen from behind (left) and from the top (right).

The bicycle is considered to have fallen when the roll angle is larger than $\frac{12\pi}{180}$ in either direction, in which case a terminal, failure state is reached, and a reward of -1 is generated. All other rewards are 0. The discount factor is $\gamma = 0.98$. The rider displacement is discretized into $\{-0.02, 0, 0.02\}$, and the torque on the handlebar into $\{-2, 0, 2\}$, leading to a discrete action space with 9 elements. This action space is sufficient to balance the bicycle.

In order to study the influence of the set of representative states on the performance of the resulting policies, two different sets of representative states are considered. The first set contains a few evenly-spaced values for the roll of the bicycle, and the rest of the state variables are zero:

$$X_{0,1} = \left\{ \tfrac{-10\pi}{180}, \tfrac{-5\pi}{180}, \ldots, \tfrac{10\pi}{180} \right\} \times \{0\} \times \{0\} \times \{0\}$$

The second set is the cross-product of a finer roll grid and a few values of the roll velocity:

$$X_{0,2} = \left\{ \tfrac{-10\pi}{180}, \tfrac{-8\pi}{180}, \ldots, \tfrac{10\pi}{180} \right\} \times \left\{ \tfrac{-30\pi}{180}, \tfrac{-15\pi}{180}, \ldots, \tfrac{30\pi}{180} \right\} \times \{0\} \times \{0\}$$

For both sets, the initial states are uniformly weighted (e.g., when $X_{0,1}$ is used, $w(x_0) = 1/|X_{0,1}| = 1/5$ for any $x_0 \in X_{0,1}$). Because we are mainly interested in the behavior of the bicycle starting from different initial rolls and roll velocities, the initial steering angle α_0 and velocity $\dot{\alpha}_0$ are always taken equal to zero; this also prevents an excessive computational cost for the CE policy search. Since a good policy can always prevent the bicycle from falling for any state in $X_{0,1}$, the optimal score (13) for this set is 0. This is no longer true for $X_{0,2}$: when ω and $\dot{\omega}$ have the same sign and are too large in magnitude, the bicycle cannot be prevented from falling by any control policy. So, the optimal score for $X_{0,2}$ is strictly negative. To prevent including in $X_{0,2}$ too many such states from which falling is unavoidable, the initial roll velocities are not taken too large in magnitude.

[10]For the bicycle and HIV examples, the measurement units of variables are mentioned only once in the text, when the variables are introduced, after which the units are omitted.

1) Balancing a deterministic bicycle: For the first set of experiments with the bicycle, the noise was eliminated from the simulations. The parameters of CE policy search were the same as for the double-integrator, i.e., $c_{\mathrm{CE}} = 10$, $\rho_{\mathrm{CE}} = 0.01$, $\alpha_{\mathrm{CE}} = 0.7$, $\varepsilon_{\mathrm{CE}} = \varepsilon_{\mathrm{MC}} = 0.001$, $d_{\mathrm{CE}} = 5$, $\tau_{\max} = 100$. Because the system is deterministic, a single trajectory was simulated from every representative state, i.e., $N_{\mathrm{MC}} = 1$. CE policy search was run while gradually increasing the number of RBFs from 3 to 8. For each set of representative states ($X_{0,1}$ and $X_{0,2}$) and every value of N, 10 independent runs were performed. The maximum number of iterations was never reached before convergence. Figure 10 presents the performance of CE policy search, and Figure 11 presents its execution time. The mean values across the 10 runs are shown, together with their 95% confidence intervals.

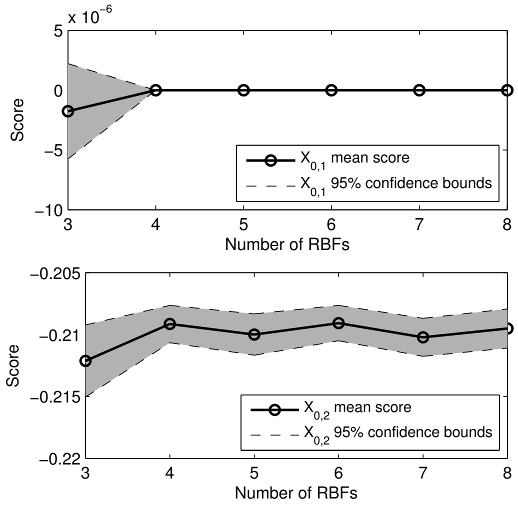

Fig. 10. Performance of CE policy search for the deterministic bicycle, using $X_{0,1}$ (top) and $X_{0,2}$ (bottom). Note the very small scale of the top figure.

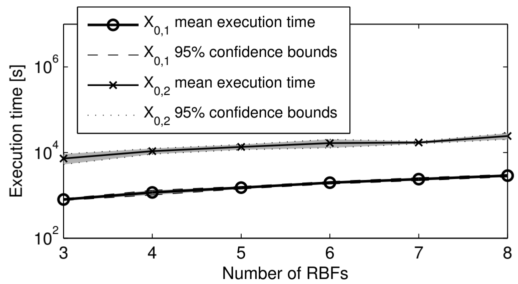

Fig. 11. Execution time of CE policy search for the deterministic bicycle.

For $X_{0,1}$ and $N \geq 4$, all the experiments reached the optimal score of 0; the scores for $N = 3$ are all extremely close to 0. For $X_{0,2}$, the performances obtained are around -0.21 and do not improve as N grows. This suggests the optimal score cannot be much larger than this value. It is remarkable that CE policy search obtains good results with as few as 3 RBFs.

The execution times of Figure 11 are comparable with those for the double integrator (Figure 4) for $X_{0,1}$, and larger for $X_{0,2}$, even though both sets contain fewer states than were considered for the double integrator. This is because simulating transitions for the bicycle requires the numerical integration of its nonlinear dynamics, which is more costly than computing the linear transitions of the double integrator.

For comparison, fuzzy Q-iteration (see Section V-A) was applied, using an equidistant grid of triangular BFs and the same 9 discrete actions. The number N' of BFs on each axis was gradually increased from 3 to 18. The resulting solutions never reached a performance comparable to CE policy search; e.g., when $N' = 18$ the score was -0.1933 for $X_{0,1}$ and -0.2383 for $X_{0,2}$. Note that $N' = 18$ led to a total of $9 \cdot 18^4 = 944784$ BFs, vastly more than the number of BFs required by CE policy search. The execution time of fuzzy Q-iteration with $N' = 18$ was 37487 s, similar to the execution time of the most computationally expensive CE policy search experiments (see Figure 11). This shows that, as the dimension of the problem increases, and also depending on the selection of representative states, CE policy search can become preferable to value-function techniques also from a computational point of view.

2) Balancing a stochastic bicycle: The second set of experiments with bicycle balancing included the effects of noise. A number of $N = 7$ RBFs was selected, and $N_{\mathrm{MC}} = 10$ trajectories were simulated from every initial state to compute the score (this number was not selected too large to prevent excessive computational costs). The rest of the parameters remained the same as in the deterministic case. For each set of representative states ($X_{0,1}$ and $X_{0,2}$), 10 independent runs were performed. The performance of the resulting policies, together with the execution time of the algorithm, are reported in Table II. The mean values across the 10 runs are shown, together with their 95% confidence intervals (in square brackets). For an easy comparison, the results in the *deterministic* case with $N = 7$ are also repeated.

TABLE II
RESULTS OF CE POLICY SEARCH FOR THE STOCHASTIC BICYCLE – COMPARISON WITH THE DETERMINISTIC CASE

	Score for $X_{0,1}$	Score for $X_{0,2}$
Stoch.	0; [0, 0]	−0.2093; [−0.2099, −0.2088]
Determ.	0; [0, 0]	−0.2102; [−0.2117, −0.2087]

	Execution time for $X_{0,1}$ [s]	Execution time for $X_{0,2}$ [s]
Stoch.	22999; [21518, 24480]	185205; [168421, 201990]
Determ.	2400; [2225, 2575]	17154; [15959, 18350]

All the scores for $X_{0,1}$ are optimal, and the scores for $X_{0,2}$ are similar to those obtained in the deterministic case. This shows that the policies computed have a good quality. The execution times are one order of magnitude larger than for the deterministic bicycle, which is expected because $N_{\mathrm{MC}} = 10$, rather than 1 as in the deterministic case.

Figure 12 illustrates the quality of two representative policies found by CE policy search. In particular, the top part of the figure shows how a policy computed using $X_{0,1}$ generalizes to initial states that do not belong to $X_{0,1}$. The bottom part similarly illustrates how a policy computed using $X_{0,2}$ generalizes. The initial states from which the policies are tested consist of a grid of values in the $(\omega, \dot{\omega})$ plane; α_0 and $\dot{\alpha}_0$ are always 0. The length of each trajectory is 50 s. This length was chosen to verify whether the bicycle is balanced robustly for a long time: it is roughly 10 times longer than the length of the trajectory used to evaluate the score during the optimization procedure, which was 5.36 s (corresponding to $K = 536$, which was computed with (6) for $\varepsilon_{\mathrm{MC}} = 0.001$). Using the larger set $X_{0,2}$ of initial states is beneficial: it leads to a policy that balances the bicycle for a much larger portion of the $(\omega, \dot{\omega})$ plane. Recall that the bicycle cannot be balanced at all from some of the states for which ω and $\dot{\omega}$ are large in magnitude and have the same sign.

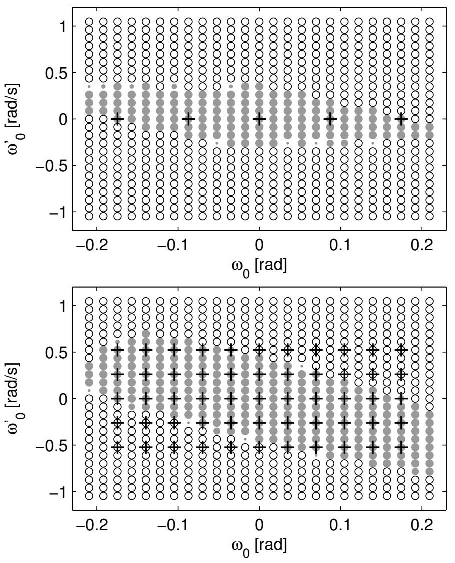

Fig. 12. Generalization of typical policies computed using $X_{0,1}$ (top) and $X_{0,2}$ (bottom). White markers indicate the bicycle was never balanced starting from that initial state; the size of the gray markers is proportional with the number of times the bicycle was properly balanced out of 10 experiments. Black crosses mark the representative states.

C. Structured treatment interruptions for HIV infection

In this section, CE policy search is used (again in simulation) to control the treatment of HIV infection. Prevalent HIV treatment strategies involve two types of drugs, called reverse transcriptase inhibitors (RTI) and protease inhibitors (PI). The negative side effects of these drugs in the long term motivate the investigation of optimal strategies for their use. One such strategy involves structured treatment interruptions (STI), where the patient is cycled on and off RTI and PI therapy, see, e.g., [36].

The HIV infection dynamics are described by a six-dimensional nonlinear model with the state vector $x = [T_1, T_2, T_1^{\mathrm{t}}, T_2^{\mathrm{t}}, V, E]^{\mathrm{T}}$, where:

- $T_1 \geq 0$ and $T_2 \geq 0$ are the counts of healthy type 1 and type 2 target cells [cells/ml].
- $T_1^{\mathrm{t}} \geq 0$ and $T_2^{\mathrm{t}} \geq 0$ are the counts of infected type 1 and type 2 target cells [cells/ml].
- $V \geq 0$ is the number of free virus copies [copies/ml].

- $E \geq 0$ is the number of immune response cells [cells/ml].

The control inputs are ε_1, the effectiveness of the RTI drug, and ε_2, the effectiveness of the PI drug. In STI, drugs are either fully administered (they are 'on'), or not at all (they are 'off'). A fully administered RTI drug corresponds to $\varepsilon_1 = 0.7$, while a fully administered PI drug corresponds to $\varepsilon_2 = 0.3$. This leads to the discrete action space $U_\text{d} = \{0, 0.7\} \times \{0, 0.3\}$. Because it is not clinically feasible to change the treatment daily, the state is measured and the drugs are switched on or off once every 5 days [36]. So, the system is controlled in discrete time with a sampling time of 5 days. For the model equations and parameters, see [36].

The system has three uncontrolled equilibria. The *uninfected* equilibrium $x_\text{n} = [1000000, 3198, 0, 0, 0, 10]^\text{T}$ is unstable: as soon as V becomes nonzero due to the introduction of virus copies, the patient becomes infected and the state drifts away from x_n. The *unhealthy* equilibrium $x_\text{u} = [163573, 5, 11945, 46, 63919, 24]^\text{T}$ is stable and represents a patient with a very low immune response, for whom the infection has reached dangerous levels. The *healthy* equilibrium $x_\text{h} = [967839, 621, 76, 6, 415, 353108]^\text{T}$ is stable and represents a patient whose immune system controls the infection without the need of drugs.

We consider the problem of using STI from the initial state x_u such that the immune response of the patient is maximized and the number of virus copies is minimized, while also penalizing the quantity of drugs administered, to account for their side effects. The reward function is [36]:

$$\rho(x, u) = -QV - R_1\varepsilon_1^2 - R_2\varepsilon_2^2 + SE \qquad (20)$$

where $Q = 0.1, R_1 = R_2 = 20000, S = 1000$. The term SE rewards the amount of immune response, $-QV$ penalizes the amount of virus copies, and $-R_1\varepsilon_1^2$, $-R_2\varepsilon_2^2$ penalize drug use.

In order to apply CE policy search, a discount factor of $\gamma = 0.99$ was used. To compute the score, the number of simulation steps was set to $K = T_\text{f}/T_\text{s}$ where $T_\text{f} = 800$ days is a sufficiently long time horizon for a good policy to control the infection [36], [37]. This leads to $K = 160$. The state variables span several orders of magnitude; to limit the effects of this large variation, a transformed state vector was used, computed as the base 10 logarithm of the original state vector. The policy was represented using $N = 8$ RBFs, and only the unhealthy initial state was used to compute the score, $X_0 = \{x_\text{u}\}$. The other parameters remained unchanged from the experiments on the other problems: $c_\text{CE} = 10$, $\rho_\text{CE} = 0.01$, $\alpha_\text{CE} = 0.7$, $\varepsilon_\text{CE} = 0.001$, $d_\text{CE} = 5$, $\tau_\text{max} = 100$.

Figure 13 shows the trajectory of the HIV system, controlled from the unhealthy initial state with the policy obtained by CE policy search. The execution time to obtain this policy was 137864 s. For comparison, trajectories obtained with no treatment and with fully effective treatment are also shown. The CE solution switches the PI drug off after approximately 300 days, but the RTI drug is left on in steady state, which means that the healthy equilibrium x_h is not reached. Nevertheless, the infection is handled much better than without STI, and the immune response E in steady state is very strong.

In the literature, x_h is reached by driving the state into the basin of attraction of this equilibrium using STI, and then switching off both drugs [36], [37]. The algorithm used in [37] automatically derives a decision-tree Q-function approximator for value iteration. This derivation produces a large number of BFs, in the order of tens of thousands or more. While our solution does not reach x_h, it still performs remarkably well. Note that, because of the high dimensionality of the HIV problem, using a value-function technique with equidistant BFs is out of the question.

VI. Conclusions and future work

In this paper, we introduced a novel algorithm for direct policy search in continuous-state, discrete-action Markov decision processes. This algorithm uses a flexible policy parametrization, inspired by the work on automatic construction of BFs for value function approximation. CE optimization is used to search for policy parameters that maximize the empirical return from a representative set of initial states. CE policy search was evaluated in a double-integrator example and in two more difficult problems: balancing an unstable bicycle, and the control of HIV infection. The algorithm reliably offered a good performance, using only a small number of BFs to represent the policy. Compared to value-function techniques with equidistant BFs, CE policy search required vastly fewer BFs to provide a good performance, and it required a larger execution time for the two-dimensional double-integrator, but a comparable execution time for the four-dimensional (deterministic) bicycle. As illustrated for HIV infection control, CE policy search can be applied to high-dimensional problems, for which value-function techniques with equidistant BFs are impractical due to excessive computational costs.

The theoretical study of CE policy search is an important opportunity for further research. Convergence results for the CE method are unfortunately only available for combinatorial optimization [15], [34], whereas CE policy search also involves the optimization of continuous variables. The convergence results for the related model-reference adaptive search [13] require the restrictive assumption that the optimal policy parameter is unique.

We applied CE policy search to optimize deterministic policies that choose among discretized actions. Nevertheless, the algorithm can be extended to stochastic or continuous-action policies, by adopting a suitable policy parametrization. For instance, the current policy parametrization can be naturally extended to continuous actions by interpolating the actions assigned to the BFs, using the BF values as weights.

In this work, CE optimization was employed, and it was shown to outperform DIRECT optimization in the double-integrator problem. It would be useful to compare CE optimization with other algorithms able to solve the global, mixed-integer, gradient-free optimization problem that arises in policy search, such as genetic algorithms, simulated annealing, and tabu search.

References

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

[2] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. MIT Press, 1998.

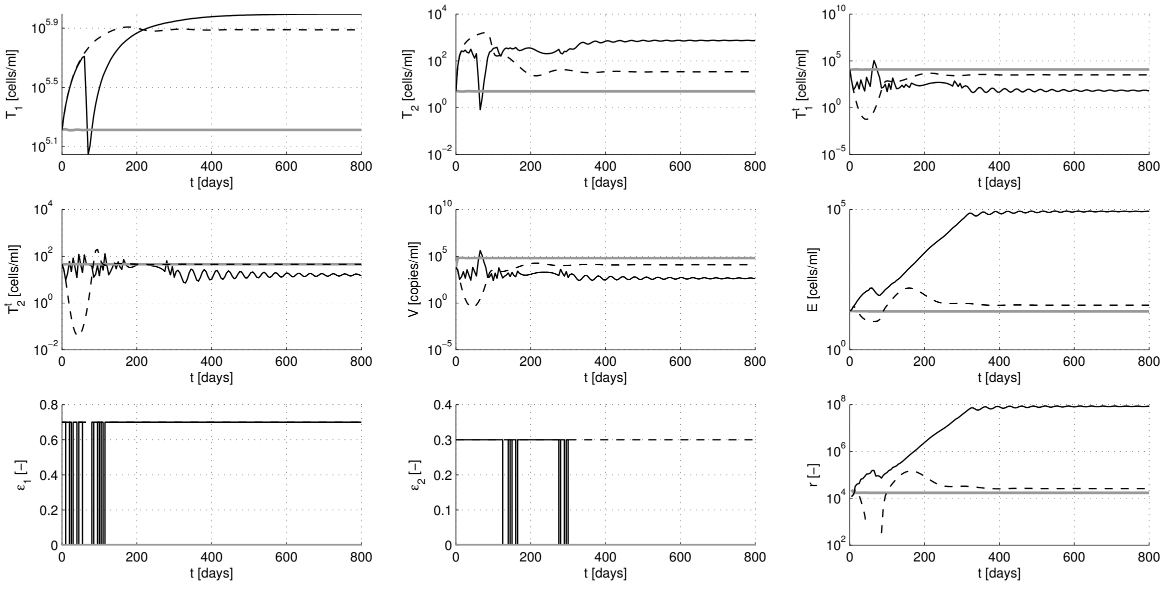

Fig. 13. Trajectories from x_u. Black, continuous: policy computed with CE policy search. Gray: no treatment. Black, dashed: fully effective treatment. The states and rewards are shown on a logarithmic scale, and negative values of the reward are ignored.

[3] D. P. Bertsekas and J. N. Tsitsiklis, *Neuro-Dynamic Programming*. Athena Scientific, 1996.

[4] J. N. Tsitsiklis and B. Van Roy, "Feature-based methods for large scale dynamic programming," *Machine Learning*, vol. 22, no. 1–3, pp. 59–94, 1996.

[5] R. Munos and A. Moore, "Variable-resolution discretization in optimal control," *Machine Learning*, vol. 49, no. 2–3, pp. 291–323, 2002.

[6] M. G. Lagoudakis and R. Parr, "Least-squares policy iteration," *Journal of Machine Learning Research*, vol. 4, pp. 1107–1149, 2003.

[7] L. Buşoniu, D. Ernst, B. De Schutter, and R. Babuška, "Continuous-state reinforcement learning with fuzzy approximation," in *Adaptive Agents and Multi-Agent Systems III*, ser. Lecture Notes in Computer Science, K. Tuyls, A. Nowé, Z. Guessoum, and D. Kudenko, Eds. Springer, 2008, vol. 4865, pp. 27–43.

[8] D. Ernst, P. Geurts, and L. Wehenkel, "Tree-based batch mode reinforcement learning," *Journal of Machine Learning Research*, vol. 6, pp. 503–556, 2005.

[9] S. Mahadevan and M. Maggioni, "Proto-value functions: A Laplacian framework for learning representation and control in Markov decision processes," *Journal of Machine Learning Research*, vol. 8, pp. 2169–2231, 2007.

[10] P. Marbach and J. N. Tsitsiklis, "Approximate gradient methods in policy-space optimization of Markov reward processes," *Discrete Event Dynamic Systems: Theory and Applications*, vol. 13, no. 1–2, pp. 111–148, 2003.

[11] S. Mannor, R. Y. Rubinstein, and Y. Gat, "The cross-entropy method for fast policy search," in *Proceedings 20th International Conference on Machine Learning (ICML-03)*, Washington, US, 21–24 August 2003, pp. 512–519.

[12] R. Munos, "Policy gradient in continuous time," *Journal of Machine Learning Research*, vol. 7, pp. 771–791, 2006.

[13] H. S. Chang, M. C. Fu, J. Hu, and S. I. Marcus, *Simulation-Based Algorithms for Markov Decision Processes*. Springer, 2007.

[14] M. Riedmiller, J. Peters, and S. Schaal, "Evaluation of policy gradient methods and variants on the cart-pole benchmark," in *Proceedings 2007 IEEE Symposium on Approximate Dynamic Programming and Reinforcement Learning (ADPRL-07)*, Honolulu, US, 1–5 April 2007, pp. 254–261.

[15] R. Y. Rubinstein and D. P. Kroese, *The Cross Entropy Method: A Unified Approach to Combinatorial Optimization, Monte-Carlo Simulation, and Machine Learning*. Springer, 2004.

[16] D. R. Jones, "DIRECT global optimization algorithm," in *Encyclopedia of Optimization*, C. A. Floudas and P. M. Pardalos, Eds. Springer, 2009, pp. 725–735.

[17] A. Y. Ng and M. I. Jordan, "PEGASUS: A policy search method for large MDPs and POMDPs," in *Proceedings 16th Conference in Uncertainty in Artificial Intelligence (UAI-00)*, Palo Alto, US, 30 June – 3 July 2000, pp. 406–415.

[18] R. S. Sutton, D. A. McAllester, S. P. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," in *Advances in Neural Information Processing Systems 12*, S. A. Solla, T. K. Leen, and K.-R. Müller, Eds. MIT Press, 2000, pp. 1057–1063.

[19] V. R. Konda and J. N. Tsitsiklis, "On actor-critic algorithms," *SIAM Journal on Control and Optimization*, vol. 42, no. 4, pp. 1143–1166, 2003.

[20] J. Peters and S. Schaal, "Natural actor-critic," *Neurocomputing*, vol. 71, no. 7–9, pp. 1180–1190, 2008.

[21] D. Liu, H. Javaherian, O. Kovalenko, and T. Huang, "Adaptive critic learning techniques for engine torque and air-fuel ratio control," *IEEE Transactions on Systems, Man, and Cybernetics—Part B: Cybernetics*, vol. 38, no. 4, pp. 988–993, 2008.

[22] H. H. Chin and A. A. Jafari, "Genetic algorithm methods for solving the best stationary policy of finite Markov decision processes," in *Proceedings 30th Southeastern Symposium on System Theory*, Morgantown, US, 8–10 March 1998, pp. 538–543.

[23] D. Barash, "A genetic search in policy space for solving Markov decision processes," in *AAAI Spring Symposium on Search Techniques for Problem Solving under Uncertainty and Incomplete Information*, Palo Alto, US, 22–24 March 1999.

[24] S.-M. Tse, Y. Liang, K.-S. Leung, K.-H. Lee, and T. S.-K. Mok, "A memetic algorithm for multiple-drug cancer chemotherapy schedule optimization," *IEEE Transactions on Systems, Man, and Cybernetics—Part B: Cybernetics*, vol. 37, no. 1, pp. 84–91, 2007.

[25] I. Menache, S. Mannor, and N. Shimkin, "Basis function adaptation in temporal difference reinforcement learning," *Annals of Operations Research*, vol. 134, no. 1, pp. 215–238, 2005.

[26] C. G. Atkeson and B. J. Stephens, "Random sampling of states in dynamic programming," *IEEE Transactions on Systems, Man, and Cybernetics—Part B: Cybernetics*, vol. 38, no. 4, pp. 924–929, 2008.

[27] M. G. Lagoudakis and R. Parr, "Reinforcement learning as classification: Leveraging modern classifiers," in *Proceedings 20th International Conference on Machine Learning (ICML-03).* Washington, US, 21–24 August 2003, pp. 424–431.

[28] D. P. Bertsekas, "Dynamic programming and suboptimal control: A survey from ADP to MPC," *European Journal of Control*, vol. 11, no. 4–5, pp. 310–334, 2005, special issue for the CDC-ECC-05 in Seville, Spain.

[29] C. Dimitrakakis and M. Lagoudakis, "Rollout sampling approximate policy iteration," *Machine Learning*, vol. 72, no. 3, pp. 157–171, 2008.

[30] L. Buşoniu, D. Ernst, B. De Schutter, and R. Babuška, "Policy search with cross-entropy optimization of basis functions," in *Proceedings 2009 IEEE International Symposium on Adaptive Dynamic Programming and*

Reinforcement Learning (ADPRL-09), Nashville, US, 30 March – 2 April 2009, pp. 153–160.

[31] J. Rust, "Numerical dynamic programming in economics," in *Handbook of Computational Economics*, H. M. Amman, D. A. Kendrick, and J. Rust, Eds. Elsevier, 1996, vol. 1, ch. 14, pp. 619–729.

[32] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: A survey," *Journal of Artificial Intelligence Research*, vol. 4, pp. 237–285, 1996.

[33] D. Ormoneit and S. Sen, "Kernel-based reinforcement learning," *Machine Learning*, vol. 49, no. 2–3, pp. 161–178, 2002.

[34] A. Costa, O. D. Jones, and D. Kroese, "Convergence properties of the cross-entropy method for discrete optimization," *Operations Research Letters*, vol. 35, no. 5, pp. 573–580, 2007.

[35] J. Randløv and P. Alstrøm, "Learning to drive a bicycle using reinforcement learning and shaping," in *Proceedings 15th International Conference on Machine Learning (ICML-98)*, Madison, US, 24–27 July 1998, pp. 463–471.

[36] B. Adams, H. Banks, H.-D. Kwon, and H. Tran, "Dynamic multidrug therapies for HIV: Optimal and STI control approaches," *Mathematical Biosciences and Engineering*, vol. 1, no. 2, pp. 223–241, 2004.

[37] D. Ernst, G.-B. Stan, J. Gonçalves, and L. Wehenkel, "Clinical data based optimal STI strategies for HIV: A reinforcement learning approach," in *Proceedings 45th IEEE Conference on Decision & Control*, San Diego, US, 13–15 December 2006, pp. 667–672.

Lucian Buşoniu is a postdoctoral fellow at the Delft Center for Systems and Control of Delft University of Technology, in the Netherlands. He received his PhD degree (*cum laude*) in 2009 from the Delft University of Technology, and his MSc degree (valedictorian) in 2003 from the Technical University of Cluj-Napoca, Romania. His current research interests include reinforcement learning and dynamic programming with function approximation, intelligent and learning techniques for control problems, and multi-agent learning.

Damien Ernst received the MSc and PhD degrees from the University of Liège in 1998 and 2003, respectively. He is currently a Research Associate of the Belgian FRS-FNRS and he is affiliated with the Systems and Modeling Research Unit of the University of Liège. Damien Ernst spent the period 2003–2006 with the University of Liège as a Postdoctoral Researcher of the FRS-FNRS and held during this period positions as visiting researcher at CMU, MIT and ETH. He spent the academic year 2006–2007 working at Supélec (France) as professor. His main research interests are in the fields of power system dynamics, optimal control, reinforcement learning, and design of dynamic treatment regimes.

Bart De Schutter is a full professor at the Delft Center for Systems and Control and at the Marine & Transport Technology department of Delft University of Technology in the Netherlands. He received the PhD degree in Applied Sciences (summa cum laude with congratulations of the examination jury) in 1996 from K.U. Leuven, Belgium. His current researcinterests include multi-agent systems, hybrid systems control, discrete-event systems, and control of intelligent transportation systems.

Robert Babuška is a full professor at the Delft Center for Systems and Control of Delft University of Technology in the Netherlands. He received his PhD degree (*cum laude*) in Control in 1997 from the Delft University of Technology, and his MSc degree (with honors) in Electrical Engineering in 1990 from Czech Technical University, Prague. His research interests include fuzzy systems modeling and identification, data-driven construction and adaptation of neuro-fuzzy systems, model-based fuzzy control and learning control. He is active in applying these techniques in robotics, mechatronics, and aerospace.