Policy Improvement: Between Black-Box Optimization and Episodic Reinforcement Learning

Freek Stulp[1,2], Olivier Sigaud[3]

[1] Robotics and Computer Vision, ENSTA-ParisTech, Paris

[2] FLOWERS Research Team, INRIA Bordeaux Sud-Ouest, Talence, France

[3] Institut des Systèmes Intelligents et de Robotique, Univ. Pierre Marie Curie CNRS UMR 7222, Paris

Abstract : Policy improvement methods seek to optimize the parameters of a policy with respect to a utility function. There are two main approaches to performing this optimization: reinforcement learning (RL) and black-box optimization (BBO). In recent years, benchmark comparisons between RL and BBO have been made, and there have been several attempts to specify which approach works best for which types of problem classes. In this article, we make several contributions to this line of research by: 1) Classifying several RL algorithms in terms of their algorithmic properties. 2) Showing how the derivation of ever more powerful RL algorithms displays a trend towards BBO. 3) Continuing this trend by applying two modifications to the state-of-the-art PI^2 algorithm, which yields an algorithm we denote PI^{BB}. We show that PI^{BB} is a BBO algorithm. 4) Demonstrating that PI^{BB} achieves similar or better performance than PI^2 on several evaluation tasks. 5) Analyzing why BBO outperforms RL on these tasks. Rather than making the case for BBO or RL – in general we expect their relative performance to depend on the task considered – we rather provide two algorithms in which such cases can be made, as the algorithms are identical in all respects *except* in being RL or BBO approaches to policy improvement.

1 Introduction

Over the last two decades, the convergence speed and robustness of policy improvement methods has increased dramatically, such that they are now able to learn a variety of challenging robotic tasks (Theodorou *et al.*, 2010; Rückstiess *et al.*, 2010b; Tamosiumaite *et al.*, 2011; Kober & Peters, 2011; Buchli *et al.*, 2011; Stulp *et al.*, 2012). Several underlying trends have accompanied this performance increase. The first is related to exploration, where there has been a transition from *action perturbing* methods, which perturb the output of the policy at each time step, to *parameter perturbing* methods, which perturb the parameters of the policy itself (Rückstiess *et al.*, 2010b). The second trend pertains to the parameter update, which has moved from *gradient-based* methods towards updates based on *reward-weighted averaging* (Stulp & Sigaud, 2012a).

A striking feature of these trends, described in detail in Section 2, is that they have moved reinforcement learning (RL) approaches to policy improvement closer and closer to black-box optimization (BBO). In fact, two state-of-the-art algorithms that have been applied to policy improvement — PI^2 (Theodorou *et al.*, 2010) and CMA-ES (Hansen & Ostermeier, 2001) — are so similar that a line-by-line comparison of the algorithms is feasible (Stulp & Sigaud, 2012a). The main difference is that whereas PI^2 is an RL algorithm — it uses information about rewards received at each time step *during* exploratory policy executions — CMA-ES is a BBO algorithm — it uses only the *total* reward received during execution, which enables it to treat the utility function as a black box that returns one scalar value.

In this article, we make the relation between RL and BBO even more explicit by taking these trends one (ultimate) step further. We do so by introducing PI^{BB} (in Section 3), which simplifies the exploration and parameter update methods of PI^2. These modifications are consistent with PI^2's derivation from stochastic optimal control. An important insight is that PI^{BB} is actually a BBO algorithm, as discussed in Section 3.3. One of the main contributions of this article is thus to draw an explicit bridge from RL to BBO approaches to policy improvement.

We thus have a pair of algorithms — PI^2 and PI^{BB}— that use the same method for exploration (parameter perturbation) and parameter updating (reward-weighted averaging), and differ only in being RL (PI^2) or

BBO (PI^{BB}) approaches to policy improvement. This opens the way to a more objective comparison of RL/BBO algorithms than, for instance, comparing eNAC and CMA-ES (Heidrich-Meisner & Igel, 2008a; Rückstiess *et al.*, 2010b), as eNAC is an action-perturbing, gradient-based RL algorithm, and CMA-ES is a parameter-perturbing, reward-weighted averaging BBO algorithm. If one is found to outperform the other on a particular task, does it do so due to the different parameter update methods? Or is it due to the difference in the policy perturbation? Or because one is an RL method and the other BBO? Using the PI^2/PI^{BB} pair allows us to specifically investigate the latter question, whilst keeping the other algorithmic features the same.

The PI^2/PI^{BB} pair may thus be the key to providing *"[s]trong empirical evidence for the power of evolutionary RL and convincing arguments why certain evolutionary algorithms are particularly well suited for certain RL problem classes"* (Heidrich-Meisner & Igel, 2008a), and could help verify or falsify the five conjectures proposed by Togelius *et al.* (2009, Section 4.1), about which types of problems are particularly suited for RL and BBO approaches to policy improvement. As a first step in this direction, we compare the performance of PI^2 and PI^{BB} in terms of convergence speed and final cost on the evaluation tasks from (Theodorou *et al.*, 2010) in Section 3. Although PI^{BB} has equal or better performance than PI^2 on these tasks, our aim in this article is not to make a case for either RL or BBO approaches to policy improvement — in general we expect the most appropriate method to vary from task to task, as discussed in Section 5 — but rather to provide a pair of algorithms that makes such targeted comparisons possible in the first place.

In summary, the main contributions of this article are: • Providing an overview and classification of policy improvement algorithms. • Deriving PI^{BB} by simplifying the perturbation and update methods of PI^2. • Empirically comparing PI^2 and PI^{BB} on the five tasks proposed by Theodorou *et al.* (2010), and showing that PI^{BB} has equal or superior performance. • Demonstrating that PI^{BB} is a BBO algorithm. In particular, it is a special case of CMA-ES. • Providing an algorithmic pair (PI^2 and PI^{BB}) that are identical *except* in being RL and BBO approaches to policy improvement, which opens the way to an objective comparison of RL and BBO.

The rest of this article is structured as follows. In the next section, we describe several policy improvement algorithms, explain their key differences, and classify them according to these differences. In Section 3, we show how PI^{BB} is derived by applying two simplifications to PI^2, and we compare the algorithms empirically on five tasks. The PI^{BB} algorithm is analyzed more closely in Section 3.3; in particular, we show that PI^{BB} is a BBO algorithm, and discuss several reasons why it outperforms PI^2 on the tasks used. In Section 6 we summarize the main contributions of the article.

2 Background

In RL, the policy π maps states to actions. The optimal policy π^* chooses the action that optimizes a cumulative reward over time. When the state and actions sets of the system are continuous, a policy cannot be represented by enumerating all actions, so parametric policy representations $\pi_{\boldsymbol{\theta}}$ are required, where $\boldsymbol{\theta}$ is a vector of parameters. Thus, finding the optimal policy π^* corresponds to finding the optimal policy parameters $\boldsymbol{\theta}^*$, i.e. those that maximize cumulative reward. As finding the $\boldsymbol{\theta}$ corresponding to the global optimum is generally too expensive, policy improvement methods are local methods that rather search for a local optimum of the expected reward.

In episodic RL, on which this article focusses, the learner executes a task until a terminal state is reached. Executing a policy from an initial state until the terminal state, called a "roll-out", leads to a trajectory $\boldsymbol{\tau}$, which contains information about the states visited, actions executed, and rewards received. Many policy improvement methods use an iterative process of exploration, where the policy is executed K times leading to trajectories $\boldsymbol{\tau}_{k=1\ldots K}$, and parameter updating, where the policy parameters $\boldsymbol{\theta}$ are updated based on this batch of trajectories. This process is explained in more detail in the generic policy improvement loop in Figure 1.

In this section, we give an overview of algorithms that implement this loop. We distinguish between three main classes of algorithms, based on whether their derivation is based mainly — they are not mutually exclusive — on principles based on lower bounds on the expected return (Section 2.1), path integral stochastic optimal control (Section 2.2) or BBO (Section 2.3).

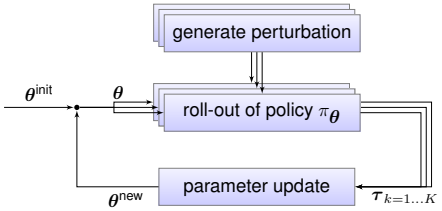

Figure 1: Generic policy improvement loop. In each iteration, the policy is executed K times. One execution of a policy is called a 'Monte Carlo roll-out', or simply 'roll-out'. Because the policy is perturbed (different perturbation methods are described in Section 2.1.3), each execution leads to slightly different trajectories in state/action space, and potentially different rewards. The exploration phase thus leads to a set of different trajectories $\boldsymbol{\tau}_{k=1\ldots K}$. Based on these trajectories, policy improvement methods then update the parameter vector $\boldsymbol{\theta} \rightarrow \boldsymbol{\theta}^{\text{new}}$ such that the policy is expected to incur lower costs/higher rewards. The process then continues with the new $\boldsymbol{\theta}^{\text{new}}$ as the basis for exploration.

2.1 Policy Improvement through Lower Bounds on the Expected Return

We now briefly describe three algorithms that build on one another to achieve ever more powerful policy improvement methods, being REINFORCE (Williams, 1992), eNAC (Peters & Schaal, 2008a), and POWER (Kober & Peters, 2011). These algorithms may be derived from a common framework based on the lower bound on the expected return, as demonstrated by Kober & Peters (2011). In this section, we focus on properties of the resulting algorithms, rather than on their derivations. Since these algorithms have already been covered in extensive surveys (Peters & Schaal, 2008a, 2007; Kober & Peters, 2011), we do not present them in full detail here, as this does not serve the particular aim of this section.

An underlying assumption of the algorithms presented in Section 2.1 and 2.2 is that the policies are represented as $\mathbf{u}_t = \mathbf{g}(\mathbf{x}, t)^\intercal \boldsymbol{\theta}$; $\mathbf{g}$ is a set of basis functions, for instance Gaussian kernels, $\boldsymbol{\theta}$ are the policy parameters, $\mathbf{x}$ is the state, and t is time since the roll-out started.

2.1.1 REINFORCE

The REINFORCE algorithm (Williams, 1992) ("reward increment = nonnegative factor $\times$ offset reinforcement $\times$ characteristic eligibility") uses a stochastic policy to foster exploration (1), where $\pi_{\boldsymbol{\theta}}(\mathbf{x})$ returns the nominal motor command[1], and $\boldsymbol{\epsilon}_t$ is a perturbation of this command at time t. In REINFORCE, this policy is executed K times with the same $\boldsymbol{\theta}$, and the states/actions/rewards that result from a roll-out are stored in a trajectory.

Given K such trajectories, the parameters $\boldsymbol{\theta}$ are then updated by first estimating the gradient $\hat{\nabla}_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$ (2) of the expected return $J(\boldsymbol{\theta}) = \mathbb{E}\left[\sum_{i=1}^{N} r_{t_i} | \pi_{\boldsymbol{\theta}}\right]$. Here, the trajectories are assumed to be of equal length, i.e. having N discrete time steps $t_{i=1\ldots N}$. The notation in (2) estimates the gradient $\hat{\nabla}_{\boldsymbol{\theta}_d} J(\boldsymbol{\theta})$ for each parameter entry d in the vector $\boldsymbol{\theta}$ separately. Riedmiller *et al.* (2007) provides a concise yet clear explanation how to derive (2). The baseline (3) is chosen so that it minimizes the variation in the gradient estimate (Peters & Schaal, 2008b). Finally, the parameters are updated through steepest gradient ascent (4), where the open parameter α is a learning rate.

Policy perturbation during a roll-out

$$\mathbf{u}_t = \pi_{\boldsymbol{\theta}}(\mathbf{x}) + \boldsymbol{\epsilon}_t \tag{1}$$

Parameter update, given K roll-outs

$$\hat{\nabla}_{\boldsymbol{\theta}_d} J(\boldsymbol{\theta}) = \frac{1}{K} \sum_{k=1}^{K} \sum_{i=1}^{N} \sum_{j=1}^{i} \nabla_{\boldsymbol{\theta}_d} \log \pi(\mathbf{u}_{t_j,k} | \mathbf{x}_{t_j,k}) (r_{t_i,k} - b_{t_i}^d) \tag{2}$$

$$b_{t_i}^d = \frac{\sum_{k=1}^{K} \sum_{j=1}^{i} \left(\nabla_{\boldsymbol{\theta}_d} \log \pi(\mathbf{u}_{t_j,k} | \mathbf{x}_{t_j,k})\right)^2 r_{t_i,k}}{\sum_{k=1}^{K} \sum_{j=1}^{i} \left(\nabla_{\boldsymbol{\theta}_d} \log \pi(\mathbf{u}_{t_j,k} | \mathbf{x}_{t_j,k})\right)^2} \tag{3}$$

$$\boldsymbol{\theta}^{new} = \boldsymbol{\theta} + \alpha \hat{\nabla}_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \tag{4}$$

[1] With this notation, the policy $\pi_{\boldsymbol{\theta}}(\mathbf{x})$ is actually deterministic. A truly stochastic policy is denoted as $\mathbf{u}_t \sim \pi_{\boldsymbol{\theta}}(\mathbf{u}|\mathbf{x}) = \mu(\mathbf{x}) + \boldsymbol{\epsilon}_t$ (Riedmiller *et al.*, 2007), where $\mu(\mathbf{x})$ is a deterministic policy that returns the nominal command. We use our notation for consistency with parameter perturbation, introduced in Section 2.1.3. For now, it is best to consider the sum $\pi_{\boldsymbol{\theta}}(\mathbf{x}) + \boldsymbol{\epsilon}_t$ to be the stochastic policy, rather than just $\pi_{\boldsymbol{\theta}}(\mathbf{x})$.

2.1.2 eNAC

One issue with REINFORCE is that it requires many roll-outs for one parameter update, and the resulting trajectories cannot be reused for later updates. This is because we need to perform a roll-out each time we want to compute $\sum_{i=1}^{N}[\dots]r_{t_i}$ in (2). Such methods are known as 'direct policy search' methods. Actor-critic methods, such as "Episodic Natural Actor Critic" (eNAC), address this issues by using a value function $V_{\pi_{\boldsymbol{\theta}}}$ as a more compact representation of long-term reward than sample episodes $R(\boldsymbol{\tau})$, allowing them to make more efficient use of samples.

In continuous state-action spaces, $V_{\pi_{\boldsymbol{\theta}}}$ cannot be represented exactly, but must be estimated from data. Actor-critic methods therefore update the parameters in two steps: 1) approximate the value function from the point-wise estimates of the cumulative rewards observed in the trajectories acquired from roll-outs of the policy; 2) update the parameters using the value function. In contrast, direct policy search updates the parameters directly using point-wise estimates[2]. The main advantage of having a value function is that it generalizes; whereas K roll-outs provide only K point-wise estimates of the cumulative reward, a value function approximated from these K point-wise estimates is also able to provide estimates not observed in the roll-outs.

Another issue is that in REINFORCE the 'naive', or 'vanilla'[3], gradient $\nabla_{\boldsymbol{\theta}}J(\boldsymbol{\theta})$ is sensitive to different scales in parameters. To find the true direction of steepest descent towards the optimum, independent of the parameter scaling, eNAC uses the Fischer information matrix F to determine the 'natural gradient': $\boldsymbol{\theta}^{new} = \boldsymbol{\theta} + \alpha F^{-1}(\boldsymbol{\theta})\nabla_{\boldsymbol{\theta}}J(\boldsymbol{\theta})$. In practice, the Fischer information matrix need not be computed explicitly (Peters & Schaal, 2008a).

Thus, going from REINFORCE to eNAC represents a transition from direct policy search to actor-critic, and from vanilla to natural gradients.

2.1.3 PoWER

REINFORCE and eNAC are both 'action perturbing' methods which perturb the nominal command at each time step $\mathbf{u}_t = \mathbf{u}_t^{\text{nominal}} + \boldsymbol{\epsilon}_t$, cf. (1). Action-perturbing algorithms have several disadvantages: 1) Samples are drawn independently from one another at each time step, which leads to a very noisy trajectory in action space (Rückstiess *et al.*, 2010b). 2) Consecutive perturbations may cancel each other and are thus washed out (Kober & Peters, 2011). The system also often acts as a low-pass filter, which further reduces the effects of perturbations that change with a high frequency. 3) On robots, high-frequency changes in actions, for instance when actions represent motor torques, may lead to dangerous behavior, or damage to the robot (Rückstiess *et al.*, 2010b). 4) It causes a large variance in parameter updates, an effect which grows with the number of time steps (Kober & Peters, 2011).

The "Policy Learning by Weighting Exploration with the Returns" (PoWER) algorithm therefore implements a different policy perturbation scheme first proposed by Rückstiess *et al.* (2010a), where the parameters $\boldsymbol{\theta}$ of the policy, rather than its output, are perturbed, i.e. $\pi_{[\boldsymbol{\theta}+\boldsymbol{\epsilon}_t]}(\mathbf{x})$ rather than $\pi_{\boldsymbol{\theta}}(\mathbf{x}) + \boldsymbol{\epsilon}_t$.

REINFORCE and eNAC estimate gradients, which is not robust when noisy, discontinuous utility functions are involved. Furthermore, they require the manual tuning of the learning rate α, which is not straightforward, but critical to the performance of the algorithms (Theodorou *et al.*, 2010; Kober & Peters, 2011). The PoWER algorithm proposed by Kober & Peters (2011) addresses these issues by using reward-weighted averaging, which rather takes a weighted average of a set of K exploration vectors $\boldsymbol{\epsilon}_{k=1\dots K}$ as follows:

Policy perturbation during a roll-out

$$\mathbf{u}_t = \pi_{[\boldsymbol{\theta}+\boldsymbol{\epsilon}_t]}(\mathbf{x}), \text{ with } \boldsymbol{\epsilon}_t \sim \mathcal{N}(0, \boldsymbol{\Sigma}) \tag{5}$$

Parameter update, given K roll-outs

$$S_{t_i}^k = \sum_{j=i}^{N} r_j^k \tag{6}$$

$$\boldsymbol{\theta}^{\text{new}} = \boldsymbol{\theta} + \left(\mathrm{E}\left[\sum_{i=1}^{N}\mathbf{W}S_{t_i}^k\right]\right)^{-1}\left(\mathrm{E}\left[\sum_{i=1}^{N}\mathbf{W}\boldsymbol{\epsilon}_{t_i}S_{t_i}^k\right]\right) \tag{7}$$

[2] The point-wise estimates are sometimes considered to be a special type of critic; in this article we use the term 'critic' only when it is a function approximator.

[3] 'Vanilla' refers to the canonical version of an entity. The origin of this expression lies in ice cream flavors; i.e. 'plain vanilla' vs. 'non-plain' flavors, such as strawberry, chocolate, etc.

$$\approx \boldsymbol{\theta} + \left(\sum_{k=1}^{K}\sum_{i=1}^{N} \mathbf{W}_{t_i} S_{t_i}^k\right)^{-1} \left(\sum_{k=1}^{K}\sum_{i=1}^{N} \mathbf{W}_{t_i} \boldsymbol{\epsilon}_{t_i}^k S_{t_i}^k\right) \tag{8}$$
$$\text{with } \mathbf{W}_{t_i} = \mathbf{g}_{t_i}\mathbf{g}_{t_i}^\intercal (\mathbf{g}_{t_i}^\intercal \boldsymbol{\Sigma} \mathbf{g}_{t_i})^{-1}$$

where K refers to the number of roll-outs, and $\mathbf{g}_{t_i}$ is a vector of the basis function activations at time t_i. The update (8) may be interpreted as taking the average of the perturbation vectors $\boldsymbol{\epsilon}_k$, but weighting them with $S_k / \sum_{l=1}^{K} S_l$, which is a normalized version of the reward-to-go S_k. Hence the name reward-weighted averaging. An important property of reward-weighted averaging is that it follows the natural gradient (Arnold *et al.*, 2011), *without* having to actually compute the gradient or the Fischer information matrix. This leads to more robust updates.

The final main difference between REINFORCE/eNAC and POWER is that the former require roll-out trajectories to contain information about the state and actions, as they must compute $\nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\mathbf{u}_t|\mathbf{x}_t)$ to perform an update. In contrast, POWER only uses information about the rewards r_t from the trajectory, as these are necessary to compute the expected return (6). The basis function $\mathbf{g}$ are parameters of the algorithm, and must not be stored in the trajectory.

In summary, going from eNAC to POWER represents a transition from action perturbation to policy parameter perturbation, from estimating the gradient to reward-weighted averaging, and from actor-critic back to direct policy search (as in REINFORCE).

2.2 Policy Improvement with Path Integrals (PI^2)

The PI^2 algorithm has its roots in path integral solutions of stochastic optimal control problems[4]. In particular, PI^2 is the application of Generalized Path Integral Control (GPIC) to parameterized policies (Theodorou *et al.*, 2010). For the complete PI^2 derivation we refer to Theodorou *et al.* (2010).

Policy perturbation during a roll-out (parameter perturbation)

$$\mathbf{u}_t = \pi_{[\boldsymbol{\theta} + \boldsymbol{\epsilon}_t]}(\mathbf{x}_t), \text{ with } \boldsymbol{\epsilon}_t \sim \mathcal{N}(0, \boldsymbol{\Sigma}) \tag{9}$$

Parameter update for each time step (reward-weighted averaging)

$$S_{t_i}^k = \phi_{t_N}^k + \sum_{j=i}^{N-1} q_{t_j}^k + \frac{1}{2}\sum_{j=i+1}^{N-1} (\boldsymbol{\theta} + \mathbf{M}_{t_j}\boldsymbol{\epsilon}_{t_j}^k)^\intercal \mathbf{R} (\boldsymbol{\theta} + \mathbf{M}_{t_j}\boldsymbol{\epsilon}_{t_j}^k) \tag{10}$$

$$P_{t_i}^k = \frac{e^{-\frac{1}{\lambda} S_{t_i}^k}}{\sum_{k=1}^{K}\left[e^{-\frac{1}{\lambda} S_{t_i}^k}\right]} \tag{11}$$

$$\delta\boldsymbol{\theta}_{t_i} = \sum_{k=1}^{K} \left[\mathbf{M}_{t_i}\ \boldsymbol{\epsilon}_{t_i}^k\ P_{t_i}^k\right] \quad \text{with } \mathbf{M}_{t_j} = \mathbf{R}^{-1}\mathbf{g}_{t_j}\ \mathbf{g}_{t_j}^T (\mathbf{g}_{t_j}^T \mathbf{R}^{-1} \mathbf{g}_{t_j})^{-1} \tag{12}$$

Weighted average over time steps

$$[\delta\boldsymbol{\theta}]_b = \frac{\sum_{i=0}^{N-1} (N-i)\ [\mathbf{w}_{t_i}]_b\ [\delta\boldsymbol{\theta}_{t_i}]_b}{\sum_{i=0}^{N-1} (N-i)\ [\mathbf{w}_{t_i}]_b} \tag{13}$$

Actual parameter update

$$\boldsymbol{\theta}^{\text{new}} = \boldsymbol{\theta} + \delta\boldsymbol{\theta} \tag{14}$$

The cost-to-go $S(\boldsymbol{\tau}_{i,k})$ is computed for each of the K roll-outs and each time step $i = 1 \ldots N$. The terminal cost ϕ_{t_N}, immediate costs q_{t_i} and command cost matrix $\mathbf{R}$ are task-dependent and provided by the user. The command cost matrix regularizes the parameter vector $\boldsymbol{\theta}$ by penalizing $\boldsymbol{\theta}^\intercal \mathbf{R} \boldsymbol{\theta}$. $\mathbf{M}_{t_j,k}$ is a projection matrix onto the range space of $\mathbf{g}_{t_j}$ under the metric $\mathbf{R}^{-1}$, cf. (Theodorou *et al.*, 2010). The weight of a roll-out $P(\boldsymbol{\tau}_{i,k})$ is computed as the normalized exponentiation of the cost-to-go. This assigns high weights to low-cost roll-outs and vice versa. In PI^2, the weight P_k is interpreted as a probability, which follows from applying the Feynman-Kac theorem to the stochastic optimal control problem, cf. (Theodorou *et al.*, 2010). The key algorithmic step is in (12), where the parameter update $\delta\boldsymbol{\theta}$ is computed for each time step i through probability weighted averaging over the exploration $\boldsymbol{\epsilon}$ of all K trials. Trajectories with higher probabilities, and thus lower costs, therefore contribute more to this parameter update, moving the parameters into regions of lower costs.

[4]Note that in the previous section, algorithms aimed at maximizing *rewards*. In optimal control, the convention is rather to define *costs*, which should be minimized.

A different parameter update $\delta\boldsymbol{\theta}_{t_i}$ is computed for each time step. To acquire one parameter vector $\boldsymbol{\theta}$, the time-dependent updates must be averaged over time, one might simply use the mean parameter vector over all time steps: $\delta\boldsymbol{\theta} = \frac{1}{N}\sum_{i=1}^{N}\delta\boldsymbol{\theta}_{t_i}$. Although temporal averaging is necessary, the particular weighting scheme used in temporal averaging does not follow from the derivation. Rather than a simple mean, Theodorou *et al.* (2010) suggest the weighting scheme in Eq. (13). It emphasizes updates earlier in the trajectory, and also makes use of the activation of the j^{th} basis function at time step i, i.e. w_{j,t_i}.

As demonstrated in (Theodorou *et al.*, 2010), PI^2 is able to outperform the previous RL algorithms for parameterized policy learning described in Section 2.1 by at least one order of magnitude in learning speed (number of roll-outs to converge) and also lower final cost performance. As an additional benefit, PI^2 has no open algorithmic parameters, except for the magnitude of the exploration noise $\boldsymbol{\Sigma}$, and the number of trials per update K.

Although having been derived from very different principles, PI^2 and PoWER use identical policy perturbation methods, and have very similar update rules. The main difference between them is that PoWER requires the cost function to behave as improper probability, i.e. be strictly positive and integrate to a constant number. This constraint can make the design of suitable cost functions more complicated (Theodorou *et al.*, 2010). For cost functions with the same optimum in PoWER's pseudo-probability formulation and PI^2 cost function, "both algorithms perform essentially identical" (Theodorou *et al.*, 2010). Another difference is that matrix inversion in PoWER's parameter update (8) is not necessary in PI^2.

2.3 Policy Improvement through Black-box Optimization

Policy improvement may also be achieved with BBO. In general, the aim of BBO is to find the solution $\mathbf{x}^* \in X$ that optimizes the objective function $J : X \mapsto \mathbb{R}$ (Arnold *et al.*, 2011). As in stochastic optimal control and PI^2, J is usually chosen to be a cost function, such that optimization corresponds to minimization. Let us highlight three aspects that define BBO: **1) Input**: no assumptions are made about the search space X; **2) Objective function**: the function J is treated as a 'black box', i.e. no assumptions are made about, for instance, its differentiability or continuity; **3) Output**: the objective function returns only one scalar value. A desirable property of BBO algorithms that are applicable to problems with these conditions is that they are able to find $\mathbf{x}^*$ using as few samples from the objective function $J(\mathbf{x})$ as possible. Many BBO algorithms, such as CEM (Rubinstein & Kroese, 2004), CMA-ES (Hansen & Ostermeier, 2001) and NES (Wierstra *et al.*, 2008), use an iterative strategy, where the current x is perturbed $\mathbf{x} + \boldsymbol{\epsilon}_{k=1\ldots K}$, and a new solution $\mathbf{x}^{\text{new}}$ is computed given the evaluations $J_{k=1\ldots K} = J(\mathbf{x} + \boldsymbol{\epsilon}_{k=1\ldots K})$.

BBO is applicable to policy improvement (Rückstiess *et al.*, 2010b) as follows: **1) Input**: the input x is interpreted as being the policy parameter vector $\boldsymbol{\theta}$. Whereas RL algorithms are tailored to leveraging the problem structure to update the policy parameters $\boldsymbol{\theta}$, BBO algorithms used for policy improvement are completely agnostic about what the parameters $\boldsymbol{\theta}$ represent; $\boldsymbol{\theta}$ might represent the policy parameters for a motion primitive to grasp an object, or simply the 2-D search space to find the minimum of a quadratic function. **2) Objective function**: the function f executes the policy π with parameters $\boldsymbol{\theta}$, and records the rewards $r_{t_{i=1:N}}$. **3) Output**: J must sum over these rewards after a policy execution: $R = \sum_{i=1}^{N} r_{t_i}$ to achieve an output of only one scalar value. Examples of applying BBO to policy improvement include (Ng & Jordan, 2000; Busoniu *et al.*, 2011; Heidrich-Meisner & Igel, 2008a; Rückstiess *et al.*, 2010b; Marin & Sigaud, 2012; Fix & Geist, 2012).

For the point of view of policy improvement, an important property of BBO algorithms is that the search is performed in the space of policy parameters; this is the input of the black-box cost function J. Therefore, all BBO approaches to policy improvement must be policy parameter perturbing methods. Furthermore, since the policy is executed *within* the function J, the parameter perturbation $\boldsymbol{\theta} + \boldsymbol{\epsilon}$ can be passed only once as an argument to J *before* the policy is executed. The perturbations $\boldsymbol{\epsilon}$ therefore cannot vary over time, as is the case in REINFORCE/eNAC/PoWER. As Heidrich-Meisner & Igel (2008a) note: "*in evolutionary strategies [BBO] there is only one initial stochastic variation per episode, while the stochastic policy introduces perturbations in every step of the episode.*".

The RL algorithms discussed in Section 2.1 (REINFORCE/eNAC/PoWER) and 2.2 (PI^2), use information about the rewards received at each time step. Furthermore, REINFORCE/eNAC also use information about the states visited, and actions executed in these states. BBO by definition cannot make use of this information, because the black-box cost function returns only a scalar cost, as visualized in Figure 2. For policy improvement, this scalar cost is taken to be the return of an episode $R = \sum_{i=1}^{N} r_{t_i}$.

In fact, *if* a policy improvement algorithm 1) uses policy perturbation, where the perturbation is constant

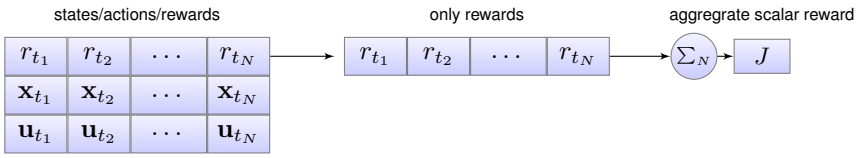

Figure 2: Illustration of the different types of information that may be stored in the trajectories that arise from policy roll-outs. Algorithms that use only the aggregate scalar reward are considered to be BBO approaches to policy improvement.

during policy execution, *and* 2) stores only the scalar total cost of a roll-out in the trajectory, *then* it is by definition a BBO approach to policy improvement, because the algorithm that works under these conditions is by definition a BBO algorithm. If a particular algorithm is historically considered to be an RL approach, we are agnostic about this. But if the two conditions above (one scalar return, constant parameter perturbation) hold, it must at least be acknowledged that the algorithm may *also* be interpreted as being a BBO algorithm. As Rückstiess *et al.* (2010b) point out: *"the return of a whole RL episode can be interpreted as a single fitness evaluation. In this case, parameter-based exploration in RL is equivalent to black-box optimization."*.

The relative advantages and disadvantages of using the states and rewards encountered during a roll-out rather than treating policy improvement as BBO depend on the domain and problem structure, and are not yet well understood (Heidrich-Meisner & Igel, 2008a).

2.3.1 BBO Algorithms Applied to Policy Improvement

CMA-ES (Hansen & Ostermeier, 2001) is an example of an existing BBO method that was applied only much later to the specific domain of policy improvement (Heidrich-Meisner & Igel, 2008a). In BBO, CMA-ES is considered to be a *de facto* standard (Rückstiess *et al.*, 2010b).

Further BBO algorithms that have been applied to policy improvement include the Cross-Entropy Method (CEM) (Rubinstein & Kroese, 2004; Busoniu *et al.*, 2011; Heidrich-Meisner & Igel, 2008b), which is very similar to CMA-ES, but has simpler methods for determining the weights P_k and performing the covariance matrix update. For a comparison of further BBO algorithms such as PGPE (Rückstiess *et al.*, 2010b) and Natural Evolution Strategies (NES) (Wierstra *et al.*, 2008) with CMA-ES and eNAC, please see the overview article by Rückstiess *et al.* (2010b).

NEAT+Q is a hybrid algorithm that actually *combines* RL and BBO in a very original way (Whiteson & Stone, 2006). Within our classification, NEAT+Q is first and foremost an actor-critic approach, as function approximation is used to explicitly represent the value function. What sets NEAT+Q apart is that the representation of the value function is evolved through BBO, with the "Neuro Evolution of Augmenting Topologies" (NEAT). This alleviates the user from having to design this representation; unsuitable representations may keep RL algorithms from being able to learn the problem, even for algorithms with proven convergence properties (Whiteson & Stone, 2006).

2.4 Classification of Policy Improvement Algorithms

Based on several of the properties of the algorithms discussed in this section, we provide a classification of these algorithms in Figure 3. The figure also depicts the aim of the next section: simplifying the PI^2 algorithm to acquire PI$^{\text{BB}}$, which is a BBO algorithm.

3 From PI^2 to PI$^{\text{BB}}$

To convert PI^2 to a BBO algorithm, we must first adapt the policy perturbation method. PI^2 and BBO both use policy parameter perturbation, but in PI^2 it varies over time ($\boldsymbol{\theta} + \boldsymbol{\epsilon}_t$), whereas it must remain constant over time in BBO ($\boldsymbol{\theta} + \boldsymbol{\epsilon}$). In Section 3.1, we therefore simplify the perturbation method in PI^2 to be constant over time, which yields the algorithm variation PI^{2-}.

Second, we must adapt PI^2 such that it is able to update the parameters based only on the scalar aggregated cost $J = \sum_{i=1}^{N} r_{t_i}$, rather than having access to the reward r_t at each time step t. This is done in Section 3.2, and yields the PI$^{\text{BB}}$ algorithm. An important aspect of these two simplifications for deriving PI$^{\text{BB}}$ from PI^2

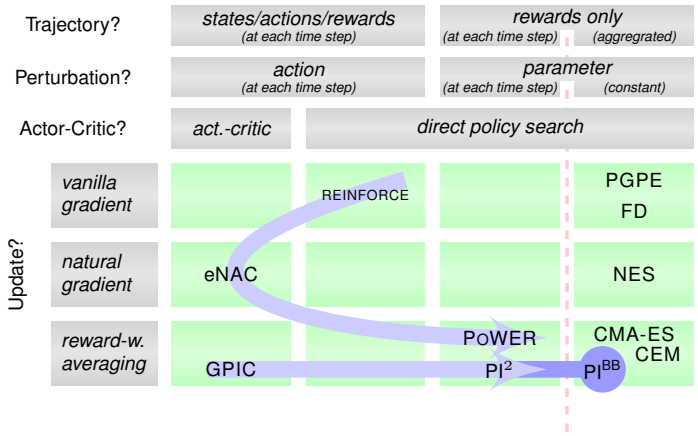

Figure 3: Classification of policy improvement algorithms. The vertical dimension categorizes the update method used, and the horizontal dimension the method used to perturb the policy. The two streams represent both the derivation history of policy improvement algorithms.

is that they do not violate any of the assumptions required for the derivation of PI^2 from stochastic optimal control, which we motivate in more detail throughout this section.

3.1 Simplifying the Exploration Method

This section is concerned with analyzing exploration in PI^2. We refer to the 'canonical' version of PI^2, which samples different exploration vectors $\boldsymbol{\epsilon}_t$ for each time step, as PI^{2}. The small blue symbol serves as a mnemonic to indicate that exploration varies at a high frequency. As an alternative to time-varying exploration, Theodorou *et al.* (2010) propose to generate exploration noise only for the basis function with the highest activation. This approach is also used by Tamosiumaite *et al.* (2011). We refer to this second method as PI^2 with exploration per basis function, or PI^{2}, where the green graphs serves as a mnemonic of the shape of the exploration for one basis function. Alternatively, $\boldsymbol{\epsilon}_{t_i,k}$ can be set to have a constant value during a roll-out (Stulp & Sigaud, 2012a). Thus, for each of the K roll-outs, we generate $\boldsymbol{\epsilon}_k$ exploration vectors before executing the policy, and keep it constant during the execution, i.e. $\boldsymbol{\epsilon}_{t_i,k} = \boldsymbol{\epsilon}_k$. We call this 'PI^2 with constant exploration', and denote it as PI^{2}, where the horizontal line indicates a constant value over time.

3.2 Simplifying the Parameter Update

In this section, we simplify the parameter update rule of PI^2 which yields the simpler PI$^{\text{BB}}$ algorithm.

In PI^2, a different parameter update $\delta\boldsymbol{\theta}_{t_i}$ is computed for each time step i. This is caused by its derivation from GPIC, where motor commands u_t are different at each time step; it is difficult to imagine a task that requires a constant motor command during a roll-out. But since the policy parameters $\boldsymbol{\theta}$ are constant during a roll-out, there is a need to condense the N parameters updates $\delta\boldsymbol{\theta}_{t_i=0:N}$ into one update $\delta\boldsymbol{\theta}$. This step is called temporal averaging, and was proposed by Theodorou *et al.* (2010) as:

$$[\delta\boldsymbol{\theta}]_d = \frac{\sum_{i=1}^{N}(N-i+1)\, w_{d,t_i}\, [\delta\boldsymbol{\theta}_{t_i}]_d}{\sum_{i=1}^{N} w_{d,t_i}(N-i+1)}. \tag{15}$$

This temporal averaging scheme emphasizes updates earlier in the trajectory, and also makes use of the basis function weights w_{d,t_i}. However, since this does not directly follow from the derivation *"[u]sers may develop other weighting schemes as more suitable to their needs."* (Theodorou *et al.*, 2010). As an alternative, we now choose a weight of 1 at the first time step, and 0 for all others. This means that all updates $\delta\boldsymbol{\theta}_{t_i}$ are ignored, except the first one $\delta\boldsymbol{\theta}_{t_1}$, which is based on the cost-to-go at the first time step $S(\boldsymbol{\tau}_{1,k})$. By definition, the cost-to-go at t_1 represents the cost of the entire trajectory. This implies that we must only compute the cost-to-go $S(\boldsymbol{\tau}_{i,k})$ and probability $P(\boldsymbol{\tau}_{i,k})$ for $i = 1$. This simplified PI^2 variant, which does not use temporal averaging, and which we denote 'PI$^{\text{BB}}$' is presented in more detail in Section 3.3.

Note that this simplification depends strongly on using constant exploration noise during a roll-out. If the noise varies at each time step or per basis function, the variation at the first time step $\boldsymbol{\epsilon}_{t_1,k}$ is not at all representative for the variations throughout the rest of the trajectory. It is therefore more accurate to consider PI$^{\text{BB}}$ as a variant of PI^{2-}, rather than of PI^2 $\equiv$PI$^{2\text{W}}$.

3.3 The PI$^{\text{BB}}$ Algorithm

PI$^{\text{BB}}$ is a special case of PI^2 in which: 1) Exploration noise is constant over time, i.e. as in PI^{2-}. 2) Temporal averaging uses only the first update, i.e. $\delta\boldsymbol{\theta}^{\text{new}} = \delta\boldsymbol{\theta}^{\text{new}}{}_{t_1}$. We also refer to this simply as 'no temporal averaging'. In this section, we perform a closer analysis of PI$^{\text{BB}}$.

Figure 4 lists both the PI^2 (left) and PI$^{\text{BB}}$ (center) algorithms. Since PI$^{\text{BB}}$ is a simplified version of PI^2, we have visualized the simplifications as dark red areas, that indicate that these lines are dropped from the PI^2 algorithm. In Figure 4, simplifications have been labeled: (X) – keep exploration constant; (T) – do not use temporal averaging; (M) – drop the projection matrix $\mathbf{M}$ from the parameter update, as motivated in (Stulp *et al.*, 2011).

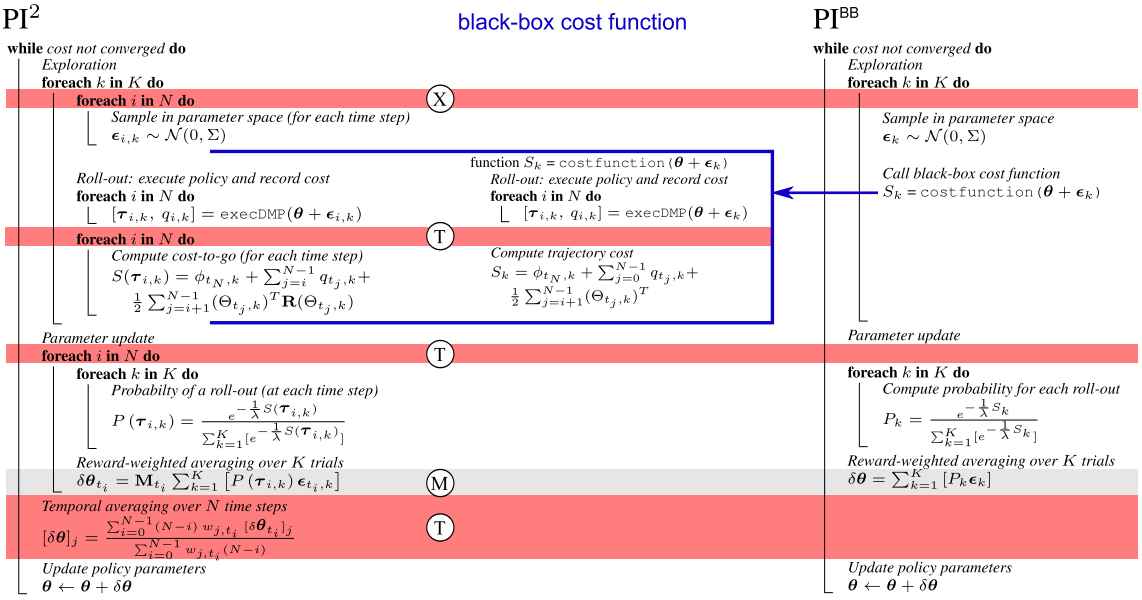

Figure 4: Comparison of PI^2 (left) and PI$^{\text{BB}}$ (right) applied to policy improvement (center). Red indicates that a line from PI^2 is no longer necessary in PI$^{\text{BB}}$.

We now demonstrate that the PI$^{\text{BB}}$ algorithm is equivalent to applying a BBO algorithm to the policy parameters. The effect of using constant exploration is that PI$^{\text{BB}}$ has only one remaining loop over time (to execute the policy), and that temporal averaging (the penultimate line of PI^2) disappears, cf. Figure 4. As a consequence, determining the cost of a vector $\boldsymbol{\theta} + \boldsymbol{\epsilon}_k$ may now be interpreted as a black-box cost function, i.e. $S_k = J(\boldsymbol{\theta} + \boldsymbol{\epsilon}_k)$ with the perturbed policy parameters (which do not vary over time due to (C1)) as input, and the scalar trajectory cost S_k as output (only the entire trajectory cost is needed due to (C2)). In PI$^{\text{BB}}$, the cost function $S_k = J(\boldsymbol{\theta} + \boldsymbol{\epsilon}_k)$ thus does the following: 1) integrate and execute the policy with constant parameters $(\boldsymbol{\theta}_k + \boldsymbol{\epsilon}_k)$; 2) record the costs at each time step during the execution; 3) when the roll-out is done, sum over the costs, and return them as S_k.

In summary, the main difference between PI^2 and PI$^{\text{BB}}$ is that PI$^{\text{BB}}$ does indeed interpret the episode as a single evaluation – and PI$^{\text{BB}}$ is thus a BBO algorithm – whereas PI^2 uses information about the cost accumulated at each time step – and PI^2 is thus a RL algorithm.

4 Empirical Comparison

The experiments in this article are based on the same tasks as presented by Theodorou *et al.* (2010), and described in Appendix A. Using these tasks, a direct comparison with the results reported by Theodorou

et al. (2010) is straightforward. Furthermore, it alleviates us of the need to demonstrate that PI^2 outperforms REINFORCE, eNAC and POWER on these tasks, because this was already done by Theodorou *et al.* (2010).

For each learning session, we are interested in comparing the convergence speed and final cost, i.e. the value to which the learning curve converges. Convergence speed is measured as the parameter update after which the cost drops below 5% of the initial cost before learning. The final cost is the mean cost over the last 100 updates. For all tasks and algorithm settings, we execute 10 learning sessions (which together we call an 'experiment'), and report the $\mu \pm \sigma$ over these 10 learning sessions. For all experiments, the Dynamic Movement Primitives (DMPs) (Ijspeert *et al.*, 2002) and PI^2 parameters are the same as in (Theodorou *et al.*, 2010), and listed in Appendix A.

Figure 5 summarizes the results of comparing the four variants of PI^2: the three different exploration methods with temporal averaging (PI$^{2\text{W}}$, PI$^{2\text{M}}$, PI^{2-}), and PI^2 without temporal averaging (PI$^{\text{BB}}$).

To evaluate the convergence speed, we determine when each of the learning curves drops below 5% of the cost before learning, as highlighted in the left graph. The means of these values for the three exploration methods are visualized as vertical lines in the left graph of Figure 5. At the top of these lines, a horizontal bar represents the standard deviation. For convergence we see that PI$^{\text{BB}}$ <PI^{2-} <PI$^{2\text{M}}$ <PI$^{2\text{W}}$ (10.4<19.7<26.3<54.9); these differences are significant (p-value < 0.001).

The right graph compares the final cost of each method, and depicts the average learning curve during the last 100 updates, after which all learning curves have converged. The vertical lines and horizontal bars in the right graphs visualize the $\mu \pm \sigma$ of the final cost over the 10 learning sessions, where the final cost is defined as the mean over a learning curve during the last 100 updates. Again, we see that PI$^{\text{BB}}$ <PI^{2-} ≈PI$^{2\text{M}}$ <PI$^{2\text{W}}$ (0.91<1.11≈1.15<1.41); these differences are significant (p-value < 0.001), except for PI^{2-} and PI$^{2\text{M}}$ (p-value $= 0.10$).

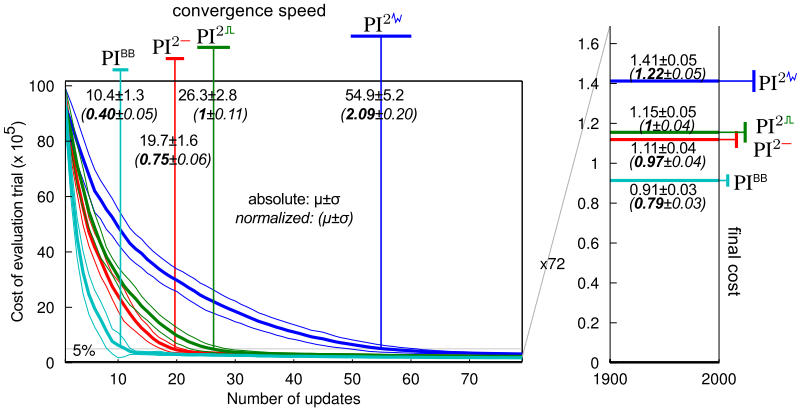

Figure 5: Learning curves for Task 4 ($\mu \pm \sigma$ over 10 learning sessions). Left: convergence speed during first 80 updates. Right: final cost during final 100 updates.

Figure 6 summarizes these results for all five tasks described in Appendix A. For each task, all values have been normalized w.r.t. the value for PI$^{2\text{M}}$, because this is the method used by Theodorou *et al.* (2010). For instance, for Task 4, the convergence below 5% of the initial cost in the bottom graph of Figure 5 was on average at updates 54.9, 26.3, 19.7, 10.4 (see bold numbers in Figure 5), which, when normalized with the result for PI$^{2\text{M}}$ (26.3), becomes 2.09, 1, 0.75, 0.40 as in Figure 6. The same is done for the final costs.

For the convergence speed (left graph), we see the same trend for each task[5]: PI$^{\text{BB}}$ <PI^{2-} <PI$^{2\text{M}}$ <PI$^{2\text{W}}$, which is significant (p-value < 0.001) except for Task 1 for PI^{2-}<$PI^{2\text{M}}$ with p-value < 0.030). Over all tasks, PI$^{\text{BB}}$ on average requires only 43% of the convergence time of PI$^{2\text{M}}$, and 57% of PI^{2-}. For the final cost (right graph), the differences between the methods are not as consistent for all tasks. However, PI$^{\text{BB}}$ leads to significantly lower costs (Task 4&5, p-value < 0.001), not significantly different (Task 1 PI$^{2\text{M}}$/PI$^{\text{BB}}$, p-value $= 0.03$), or the same (Task 2&3, all methods lead to a final cost of 0).

In summary, PI$^{\text{BB}}$ converges significantly (p-value<0.001) and substantially (1.75 = 1/0.57) faster than all exploration variants of PI^2. Furthermore, it leads to lower or the same similar final costs, depending on the task.

[5]Part of this trend may be explained by a scale factor between the different exploration methods; see (Stulp & Sigaud, 2012b)

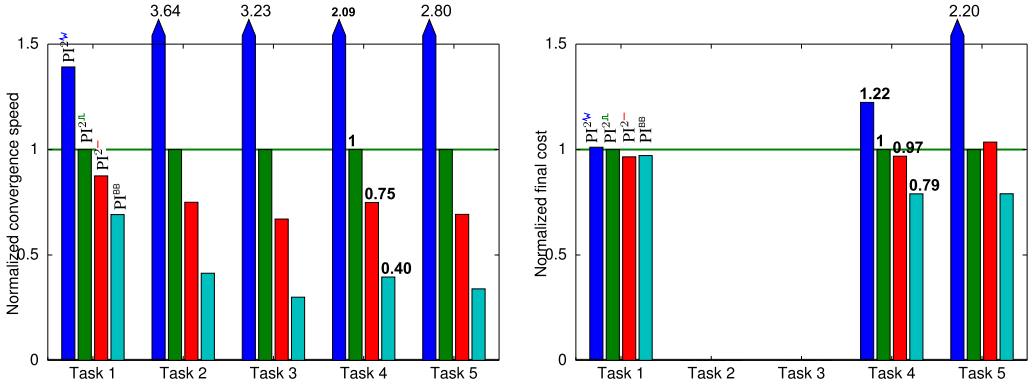

Figure 6: Results for all tasks, normalized w.r.t. $\text{PI}^{2.\text{fi}}$. Left: convergence speed. Right: final cost.

5 Discussion

So why, on these five tasks, is PI^2 outperformed by the much simpler BBO algorithm PI^{BB}? It is rather counter-intuitive that an algorithm that uses *less* information is able to converge as fast or quicker than an algorithm that uses more information. This intuition is captured well in the following quote from Moriarty *et al.* (1999) *"In this sense, EA [BBO] methods pay less attention to individual decisions than TD [RL] methods do. While at first glance, this approach appears to make less efficient use of information, it may in fact provide a robust path toward learning good policies."*

In this discussion section, we first describe previous comparisons of RL and BBO algorithms, and then explain how our results extend the knowledge obtained in this previous work. In particular, we re-consider the trends in Figure 3. We also discuss how the results we have obtained are influenced by the choice of policy representation and tasks used by Theodorou *et al.* (2010) and ourselves, which are tailored to the domain of learning skills on physical robots.

5.1 Previous Work on Empirically Comparing RL and BBO

The earliest empirical comparison of RL and BBO that we are aware of is the work of Moriarty *et al.* (1999). They compare "Evolutionary Algorithms for Reinforcement Learning" (EARL) with Q-learning on a simple MDP grid world, and conclude that these two methods "while complementary approaches, are by no means mutually exclusive." and that BBO approaches are advantageous "in situations where the sensors are inadequate to observe the true state of the world." (Moriarty *et al.*, 1999).

Heidrich-Meisner & Igel (2008b) compare the performance of CMA-ES and NAC on a single pole balancing task. They conclude that *"Our preliminary comparisons indicate that the* CMA-ES *is more robust w.r.t. to the choice of hyperparameters and initial policies. In terms of learning speed, the natural policy gradient ascent performs on par for fine-tuning and may be preferable in this scenario."* This work was later extended to a double pole balancing task (Heidrich-Meisner & Igel, 2008a), where similar conclusions were drawn. A more extensive evaluation on single- and double pole balancing tasks is performed by (Gomez *et al.*, 2008). They also conclude that *"in real world control problems, neuroevolution [. . .] can solve these problems much more reliably and efficiently than non-evolutionary reinforcement learning approaches"*.

In an extensive comparison, Rückstiess *et al.* (2010b) compare PGPE, REINFORCE, NES, eNAC, NES and CMA-ES on pole-balancing, biped standing, object grasping, and ball catching. Their focus is particularly on comparing action-perturbing and parameter-perturbing algorithms. Their main conclusion is that parameter-perturbation outperforms action-perturbation: *"We believe that parameter-based exploration should play a more important role not only for [policy gradient] methods but for continuous RL in general, and continuous value-based RL in particular"* (Rückstiess *et al.*, 2010b).

An issue with such comparisons is that *"each of these efforts typically only compares a few algorithms on a single problem, leading to contradictory results regarding the merits of different RL methods."* (Togelius *et al.*, 2009). It is this issue that we referred to in the introduction: if CMA-ES outperforms eNAC on a particular task, is it because of their different perturbation methods, their different parameter update methods, or because one is BBO and the other is RL? One of the main results of this article is to provide an algorithmic framework that allows us to specifically investigate the latter question, whilst keeping the other

algorithmic features the same.

Kalyanakrishnan & Stone (2011) aim at *"characterizing reinforcement learning methods through parameterized learning problems"*. They compare Sarsa, ExpSarsa, Q-learning, CEM and CMA-ES on problems consisting of simple square grids with a finite number of states. One interesting conclusion is that they are able to partially corroborate several of the conjectures by Togelius *et al.* (2009). The main difference to previous work is that *"our parameterized learning problem enables us to evaluate the effects of individual parameters while keeping others fixed."* (Kalyanakrishnan & Stone, 2011). Our work is orthogonal to this, in that it provides a pair of algorithms in which experimenters may switch between BBO and RL, whilst keeping other algorithmic features fixed. We thus focus on 'parameterizable algorithms', rather than parameterizable learning problems.

5.2 Reconsidering the Observed Trends

We now reconsider and evaluate the trends in Figure 3, given the empirical results presented in this article.

From Gradient-based Methods to Reward-weighted averaging. We believe the trend from gradient-based methods to reward-weighted averaging to have been an important step in enabling policy improvement methods to become robust towards noisy, discontinuous cost functions. From a theoretical perspective, we find it striking that reward-weighted averaging may be derived from fundamental principles in a wide variety of domains: reinforcement learning (Kober & Peters, 2011), stochastic optimal control (Theodorou *et al.*, 2010), rare-event theory (Rubinstein & Kroese, 2004), and a basic set of optimality principles in BBO (Arnold *et al.*, 2011). In practice, two algorithms that use this principle in BBO (CMA-ES) and RL (PI^2) turn out to be state-of-the-art in terms of empirical performance.

Previous work has focussed on comparing BBO methods that use reward-weighted averaging with gradient-based RL methods (Heidrich-Meisner & Igel, 2008a,b; Rückstiess *et al.*, 2010b). However, more recent RL methods based on reward-weighted averaging — such as POWER and PI^2— have been shown to substantially outperform gradient-based RL methods (Kober & Peters, 2011; Theodorou *et al.*, 2010). Therefore, the superiority of reward-weighted BBO over gradient-based RL on a specific task might be rather due to the update method, rather than whether BBO or RL is being used. The reason why such comparisons have not yet been made is that POWER and PI^2 have only been introduced recently. To the best of our knowledge, this work is the first in comparing RL and BBO algorithms that are both based on reward-weighted averaging. Our conclusion is that BBO (PI^{BB}) is still able to outperform reward-weighted averaging RL (PI^2) on the tasks considered, but the margin is much smaller than when comparing BBO with gradient-based RL (e.g. eNAC). We expect this margin to increase again when using more sophisticated BBO algorithms, such as CMA-ES, than PI^{BB}. This is part of our current work.

From Action Perturbation to Parameter Perturbation. Going from action perturbation to parameter perturbation seems to have been a fruitful trend, as confirmed by Rückstiess *et al.* (2010b). Mapping action perturbations to parameter updates requires a mapping from action space to parameter space, and requires knowledge of the derivative of the policy. In contrast, parameter perturbing methods perform exploration in the same space as in which the parameter update takes place. Thus, the controlled variable that leads to variations in the cost is also directly the variable that will be updated. Empirically, algorithms based on parameter perturbation substantially outperform those based on action perturbation (Rückstiess *et al.*, 2010b; Heidrich-Meisner & Igel, 2008a; Theodorou *et al.*, 2010).

From Rewards at Each Time Step to Aggregrated Costs. Although PI^{BB}, which uses only a scalar aggregrated cost, outperforms PI^2, which uses the costs at each time step, on the tasks presented by Theodorou *et al.* (2010) and used in this article, we do not hold this to be a general result. We believe the cause must lie in the chosen tasks themselves, or the particular policy representation we have chosen. These tasks and representations in their turn are biased by our particular interest in applying policy improvement to acquire robotic skills. In fact, we have (informally) compared PI^{BB} and PI^2 on several other robotic tasks not reported here, and have not found one instance where PI^2 outperforms PI^{BB}. Thus, understanding why BBO outperforms RL on these types of tasks may be related to understanding if and how the properties of typical tasks and policy representations used in robotic skill learning make them particularly amenable to BBO. According to Kalyanakrishnan & Stone (2011): *"[T]he relationships between problem instances and the performance properties of algorithms are unclear, it becomes a worthwhile pursuit to uncover them"*. The results presented in this article are a first step in the pursuit to uncover the relationship between typical robotic tasks and the performance properties of BBO/RL. This topic, of particular interest to roboticists, is at the center of our current investigations.

6 Conclusion

We have provided a classification of policy improvement algorithms. Furthermore, within this classification, we observed three trends in the chronology and derivation paths of algorithms: from gradient-based methods to reward-weighted averaging, from action to parameter perturbation, and towards algorithms that use only reward information from policy roll-outs. We observe that these trends have brought RL methods closer and closer to BBO approaches to policy improvement.

In previous work, it was shown that PI^2 is able to outperform PEGASUS, REINFORCE, eNAC and POWER (Theodorou *et al.*, 2010). Using exactly the same tasks, we observe rather surprisingly that the much simpler BBO algorithm PI^{BB} has equal or better performance than PI^2 still. Previous work on comparing RL and BBO shows that BBO often wins by a wide margin; the caveat being that, in those experiments, the BBO methods use reward-weighted averaging whereas the RL methods use gradient estimation. An important conclusion of our results is that the margin, though still existent, is much smaller when *both* RL and BBO are based on the powerful concept of reward-weighted averaging.

But rather than making the case for BBO or RL, one of the main contributions of this article is to provide an algorithmic framework in which such cases may be made. Because PI^{BB} and PI^2 use identical perturbation and parameter update methods, and differ only in being BBO and RL approaches respectively, this facilitates a more objective comparison of BBO and RL than for instance comparing algorithms that differ in many respects. Therefore, we believe this algorithmic pair is an excellent basis for comparing BBO and RL approaches to policy improvement, and further investigating the five conjectures in (Togelius *et al.*, 2009).

Although BBO thus trumps RL on the tasks we consider, we do not believe this to be a general result, and further investigations are needed, especially into the bias that typical tasks and policy representations used in robotics – the types used in this article – introduce into RL problems.

References

ARNOLD L., AUGER A., HANSEN N. & OLLIVIER Y. (2011). *Information-Geometric Optimization Algorithms: A Unifying Picture via Invariance Principles*. Rapport interne, INRIA Saclay.

BUCHLI J., STULP F., THEODOROU E. & SCHAAL S. (2011). Learning variable impedance control. *International Journal of Robotics Research*, **30**(7), 820–833.

BUSONIU L., ERNST D., SCHUTTER B. D. & BABUSKA R. (2011). Cross-entropy optimization of control policies with adaptive basis functions. *IEEE Transactions on Systems, Man, and Cybernetics-Part B: Cybernetics*, **41**(1), 196–209.

FIX J. & GEIST M. (2012). Monte-carlo swarm policy search. In *Symposium on Swarm Intelligence and Differential Evolution*, Lecture Notes in Artificial Intelligence (LNAI), p. 9 pages. Zakopane (Poland): Springer Verlag - Heidelberg Berlin.

GOMEZ F., SCHMIDHUBER J. & MIIKKULAINEN R. (2008). Accelerated neural evolution through cooperatively coevolved synapses. *Journal of Machine Learning Research*, **9**, 937–965.

HANSEN N. & OSTERMEIER A. (2001). Completely derandomized self-adaptation in evolution strategies. *Evolutionary Computation*, **9**(2), 159–195.

HEIDRICH-MEISNER V. & IGEL C. (2008a). Evolution strategies for direct policy search. In *Proceedings of the 10th international conference on Parallel Problem Solving from Nature: PPSN X*, p. 428–437, Berlin, Heidelberg: Springer-Verlag.

HEIDRICH-MEISNER V. & IGEL C. (2008b). Similarities and differences between policy gradient methods and evolution strategies. In *ESANN 2008, 16th European Symposium on Artificial Neural Networks, Bruges, Belgium, April 23-25, 2008, Proceedings*, p. 149–154.

IJSPEERT A. J., NAKANISHI J. & SCHAAL S. (2002). Movement imitation with nonlinear dynamical systems in humanoid robots. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*.

KALYANAKRISHNAN S. & STONE P. (2011). Characterizing reinforcement learning methods through parameterized learning problems. *Machine Learning*, **84**(1-2), 205–247.

KOBER J. & PETERS J. (2011). Policy search for motor primitives in robotics. *Machine Learning*, **84**, 171–203.

MARIN D. & SIGAUD O. (2012). Towards fast and adaptive optimal control policies for robots: A direct policy search approach. In *Proceedings Robotica*, p. 21–26, Guimaraes, Portugal.

MORIARTY D. E., SCHULTZ A. C. & GREFENSTETTE J. J. (1999). Evolutionary algorithms for reinforcement learning. *Journal of Artificial Intelligence (JAIR)*, **11**, 241–276.

NG A. Y. & JORDAN M. I. (2000). Pegasus: A policy search method for large mdps and pomdps. In *Proceedings of the 16th Conference on Uncertainty in Artificial Intelligence*, p. 406–415.

PETERS J. & SCHAAL S. (2007). Applying the episodic natural actor-critic architecture to motor primitive learning. In *Proceedings of the 15th European Symposium on Artificial Neural Networks (ESANN 2007)*, p. 1–6.

PETERS J. & SCHAAL S. (2008a). Natural actor-critic. *Neurocomputing*, **71**(7-9), 1180–1190.

PETERS J. & SCHAAL S. (2008b). Reinforcement learning of motor skills with policy gradients. *Neural networks : the official journal of the International Neural Network Society*, **21**(4), 682–97.

POWELL W. B. (2007). *Approximate Dynamic Programming: Solving the curses of dimensionality*, volume 703. Wiley-Blackwell.

RIEDMILLER M., PETERS J. & SCHAAL S. (2007). Evaluation of Policy Gradient Methods and Variants on the Cart-Pole Benchmark. In *2007 IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning*, p. 254–261: IEEE.

RUBINSTEIN R. & KROESE D. (2004). *The Cross-Entropy Method: A Unified Approach to Combinatorial Optimization, Monte-Carlo Simulation, and Machine Learning*. Springer-Verlag.

RÜCKSTIESS T., FELDER M. & SCHMIDHUBER J. (2010a). State-dependent exploration for policy gradient methods. In *19th European Conference on Machine Learning (ECML)*.

RÜCKSTIESS T., SEHNKE F., SCHAUL T., WIERSTRA D., SUN Y. & SCHMIDHUBER J. (2010b). Exploring parameter space in reinforcement learning. *Paladyn. Journal of Behavioral Robotics*, **1**, 14–24.

SANTAMARÍA J., SUTTON R. & RAM A. (1997). Experiments with reinforcement learning in problems with continuous state and action spaces. *Adaptive behavior*, **6**(2), 163–217.

STULP F. & SIGAUD O. (2012a). Path integral policy improvement with covariance matrix adaptation. In *Proceedings of the 29th International Conference on Machine Learning (ICML)*.

STULP F. & SIGAUD O. (2012b). Policy improvement methods: Between black-box optimization and episodic reinforcement learning. hal-00738463.

STULP F., THEODOROU E., KALAKRISHNAN M., PASTOR P., RIGHETTI L. & SCHAAL S. (2011). Learning motion primitive goals for robust manipulation. In *International Conference on Intelligent Robots and Systems (IROS)*.

STULP F., THEODOROU E. & SCHAAL S. (2012). Reinforcement learning with sequences of motion primitives for robust manipulation. *IEEE Transactions on Robotics*, **28**(6), 1360–1370.

SUTTON R. & BARTO A. (1998). *Reinforcement Learning: an Introduction*. MIT Press.

TAMOSIUMAITE M., NEMEC B., UDE A. & WÖRGÖTTER F. (2011). Learning to pour with a robot arm combining goal and shape learning for dynamic movement primitives. *Robots and Autonomous Systems*, **59**(11), 910–922.

THEODOROU E., BUCHLI J. & SCHAAL S. (2010). A generalized path integral control approach to reinforcement learning. *Journal of Machine Learning Research*, **11**, 3137–3181.

TOGELIUS J., SCHAUL T., WIERSTRA D., IGEL C., GOMEZ F. & SCHMIDHUBER J. (2009). Ontogenetic and phylogenetic reinforcement learning. *Zeitschrift Künstliche Intelligenz - Special Issue on Reinforcement Learning*, p. 30–33.

WHITESON S. & STONE P. (2006). Evolutionary function approximation for reinforcement learning. *Journal of Machine Learning Research*, **7**, 877–917.

WIERSTRA D., SCHAUL T., PETERS J. & SCHMIDHUBER J. (2008). Natural evolution strategies. In *Proceedings of IEEE Congress on Evolutionary Computation (CEC)*.

WILLIAMS R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, **8**, 229–256.

A Evaluation Tasks

We describe the tasks used for the empirical evaluations in Section 4. The implementations are based on the same source code as used for Theodorou *et al.* (2010), and all tasks and algorithms parameters are the same unless stated otherwise. This allows for a direct comparison of the results in this article and those acquired by Theodorou *et al.* (2010). Due to the similarity, this appendix is very similar to Section 5 of (Theodorou *et al.*, 2010), and added for completeness only.

Parameterizations. As in (Theodorou *et al.*, 2010), we use Dynamic Movement Primitives (Ijspeert *et al.*, 2002) as the underlying policy representation in all tasks. The DMPs have 10 basis functions per dimension, and a duration

of 0.5s. During learning, $K = 15$ roll-outs are performed for one update. Although 10 roll-outs has usually proven to be sufficient, Theodorou *et al.* (2010) choose 15 roll-outs to allow comparison with eNAC, which requires at least 1 roll-out more than the number of basis functions to perform its matrix inversion without numerical instabilities. The initial exploration magnitude is $\lambda = 0.05$ for all tasks except Task 1, where it is $\lambda = 0.01$. The exploration decay is $\gamma = 0.99$, i.e. the exploration at update u is $\mathbf{\Sigma}_u = \gamma^u \lambda \mathbf{I}$.

Task 1. This task considers a 1-dimensional DMP of duration 0.5s, which starts at $x_0 = 0$ and ends at the goal $g = 1$. In this task as in all others, the initial movement is acquired by training the DMP with a minimum-jerk movement. The aim of Task 1 is to reach the goal g with high accuracy, whilst minimizing acceleration, which is expressed with the following immediate (r_t) and terminal (ϕ_{t_N}) costs:

$$r_t = 0.5 f_t^2 + 5000 \boldsymbol{\theta}^\intercal \boldsymbol{\theta}, \quad \phi_{t_N} = 10000(\dot{x}_{t_N}^2 + 10(g - x_{t_N})^2) \tag{16}$$

where f_t refers to the linear spring-damper system in the DMP (Theodorou *et al.*, 2010). Figure 7 (left) visualizes the movement before and after learning.

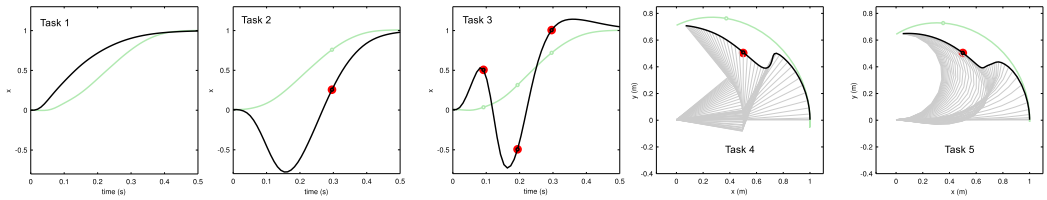

Figure 7: Task 1: Reaching the goal accurately whilst minimizing accelerations before (light green) and after (black) learning. Task 2 and 3: Minimizing the distance to 1 or 3 viapoints before (light green) and after (black) learning. Task 4 (2-DOF) and 5 (10-DOF): minimizing the distance to a viapoint in end-effector space whilst minimizing joint accelerations.

Task 2 & 3. In Task 2, the aim is for the output of the 1-dimensional DMP (same parameters as in Task 1) to pass through the viapoint 0.25 at time $t = 300ms$. Which is expressed with the costs:

$$r_{300ms} = 10^8 (0.25 - x_{t_{300ms}})^2, \quad \phi_{t_N} = 0 \tag{17}$$

The costs are thus 0 at each time step except at t_{300ms}. This cost function was chosen by Theodorou *et al.* (2010) to allow for the design of a compatible function for POWER.

Task 3 is equivalent except that it uses 3 viapoints [0.5 -0.5 1.0] at times [100ms 200ms 300ms] respectively. Figure 7 (2nd and 3rd graphs) visualizes the movement before and after learning for Task 2 and Task 3.

Note that Task 3 was not evaluated by Theodorou *et al.* (2010). We have included it as we expected that it is a task where it may be *"helpful to exploit intermediate rewards"* (Togelius *et al.*, 2009), and where RL approaches are conjectured to outperform BBO (Togelius *et al.*, 2009). As Figure 6 reveals, this is not the case for this particular task, and PI^{BB} also outperforms PI^2 for this task.

Task 4 & 5. Theodorou *et al.* (2010) used this task to evaluate the scalability of PI^2 to high-dimensional action spaces and learning problems with high redundancy. Here, an 'arm' with D rotational joints and D links of length $\frac{1}{D}$ is kinematically simulated in 2D Cartesian space. Figure 7 (right two graphs) visualizes the movement by showing the configuration of the arm at each time step. The goal is again to pass through a viapoint (0.5,0.5) , this time in end-effector space, whilst minimizing accelerations. The D joint trajectories are initialized with a minimum-jerk trajectory, and then optimized with respect to the following cost function:

$$r_t = \frac{\sum_{i=1}^{D} (D + 1 - i)(0.1 f_{i,t}^2 + 0.5 \boldsymbol{\theta}_i^\intercal \boldsymbol{\theta}_i)}{\sum_{j=1}^{D} (D + 1 - j)} \tag{18}$$

$$r_{300ms} = 10^8 ((0.5 - x_{t_{300ms}})^2 + (0.5 - y_{t_{300ms}})^2) \tag{19}$$

$$\phi_{t_N} = 0 \tag{20}$$

The weighting term $(D+1-i)$ places more weight on proximal joints than distal ones, which is motivated by the fact that proximal joints have lower mass and therefore less inertia, and are therefore more efficient to move (Theodorou *et al.*, 2010). Figure 7 depicts the movements before and after learning for arms with $D = 2$ and $D = 10$ links respectively.