Learning Step-Size Adaptation in CMA-ES

Gresa Shala[1⋆], André Biedenkapp[1⋆], Noor Awad[1], Steven Adriaensen[1], Marius Lindauer[2], and Frank Hutter[1,3]

[1] University of Freiburg, Germany
{shalag, biedenka, awad, adriaens, fh}@cs.uni-freiburg.de
[2] Leibniz University Hannover, Germany
lindauer@tnt.uni-hannover.de
[3] Bosch Center for Artificial Intelligence

Abstract. An algorithm's parameter setting often affects its ability to solve a given problem, e.g., population-size, mutation-rate or crossover-rate of an evolutionary algorithm. Furthermore, some parameters have to be adjusted dynamically, such as lowering the mutation-strength over time. While hand-crafted heuristics offer a way to fine-tune and dynamically configure these parameters, their design is tedious, time-consuming and typically involves analyzing the algorithm's behavior on simple problems that may not be representative for those that arise in practice. In this paper, we show that formulating dynamic algorithm configuration as a reinforcement learning problem allows us to automatically learn policies that can dynamically configure the mutation step-size parameter of Covariance Matrix Adaptation Evolution Strategy (CMA-ES). We evaluate our approach on a wide range of black-box optimization problems, and show that (i) learning step-size policies has the potential to improve the performance of CMA-ES; (ii) learned step-size policies can outperform the default Cumulative Step-Size Adaptation of CMA-ES; and transferring the policies to (iii) different function classes and to (iv) higher dimensions is also possible.

Keywords: Evolutionary Algorithms · Reinforcement Learning · Algorithm Configuration.

1 Introduction

Designing algorithms requires careful design of multiple components. Having the foresight of how these components will interact for all possible applications is an infeasible task. Therefore, instead of hard-wiring algorithms, human developers often expose difficult design decisions as parameters of the algorithm [26]. To make the algorithm usable off-the-shelf, they provide a default configuration that is a myopic compromise for different use-cases and often leads to sub-optimal performance on new applications.

Automated algorithm configuration can alleviate users from the burden of having to manually configure an algorithm and exceeds human performance in

⋆ Equal Contribution

a wide variety of domains [7, 27, 43, 42, 5, 29]. One shortcoming, however, is that the learned configuration is static. In practice, many algorithms are of an iterative nature and might require different parameter configurations at different stages of their execution. In evolutionary algorithms this kind of "parameter control" is often achieved through so-called self-adaptive mechanisms [9, 34, 2]. Based on some statistics of the algorithm's behavior, self-adaptation adjusts the parameter on-the-fly and thereby directly influences the algorithm's execution.

Similarly in the well-known CMA-ES [19] the step-size is adapted based on the observed evolution path by a handcrafted heuristic, called CSA [25]. Through this step-size control, CMA-ES is able to avoid premature convergence of the population [21]. However, designing heuristics to adapt not only over a time-horizon but also to the task at hand is more difficult than to simply expose the parameters and configure them at every step.

In this work, we aim to strike a balance between self-adaptive mechanisms and automated algorithm configuration by making use of dynamic algorithm configuration (DAC) [10]. Instead of only learning the optimal *initial* step-size and adapting that by a handcrafted heuristic throughout the run of the algorithm, we learn a DAC policy in a fully automatic and data-driven way that determines how the step-size should be adjusted during the CMA-ES execution.

To learn DAC policies, we make use of guided policy search (GPS) [37], a commonly used reinforcement learning (RL) technique, originating from the robotics community, capable of learning complex non-linear policies from fairly few trials. Our choice for this particular method was motivated by its capability to learn simple first-order optimizers from scratch [39]. An appealing feature of GPS is that it allows us to employ known adaptation schemes as teacher mechanism to warm-start the search. This learning paradigm allows the agent to simply imitate the teacher if it was already optimal for a specific problem, while learning to do better in areas where the teacher struggled to perform well.

We study the potential of this DAC approach to step-size adaptation in CMA-ES for a variety of black-box optimization problems. One important open question so far is how such data-driven approaches can generalize to different settings (e.g., longer optimization runs, higher-dimensional problems or different problem classes) that were not observed during training. More specifically, our contributions are:

1. We address the problem of learning step-size control for CMA-ES from a reinforcement learning perspective;
2. We propose how to model the state space, action space and reward function;
3. To use guided policy search for learning a DAC policy in efficient way, we propose to use a strong teacher guidance.
4. We empirically demonstrate that our learned DAC policies are able to outperform CMA-ES' handcrafted step-size adaptation;
5. We demonstrate the generality of our DAC approach by transferring the learned policies to (i) functions of higher dimensions, (ii) unseen test function and (iii) to a certain degree to longer optimization trajectories.

2 Related Work

Parameter Control using Reinforcement Learning The potential generality of DAC via RL is widely recognized [33, 1, 10] and RL has been applied to various specific parameter control settings [45, 46, 12, 15, 49, 8, 18, 33, 51]. However, RL covers a wide variety of techniques, and our methodology differs from prior-art in the area, in two important ways. First, GPS learns configuration policies *offline*, while most previous research considers the online setting. They attempt to learn how to adapt the parameters of an algorithm "while it is being used", i.e. without separate training phase. While desirable, online learning introduces a challenging exploration-exploitation trade-off. Also, experience is typically not transferred across runs, similar to hand-crafted adaptation mechanisms. That being said, prior-art considering the offline setting does exist, e.g., Battiti et al. [8] for local search SAT solvers and Sharma et al. [51] for EA. Second, GPS belongs to the family of *policy search* methods, which are often able to handle partially observable state spaces and continuous actions spaces better than previously used value-based RL methods.

Black-Box Dynamic Algorithm Configuration In a sense, our methodology more closely resembles static algorithm configuration (AC) than traditional RL approaches. We represent configuration policies as a neural network; and as in AC, train it offline. Instead of GPS, black-box optimizers, e.g. ES, could also be used to optimize these weights [31, 50, 17]. In fact, prior-art exists that performs DAC using static AC methods [35, 3, 6, 32]. A limitation of these "black-box" approaches is that they are unaware of the dynamic nature of the problem [1], e.g. which configurations where used at each time step, and how this affected execution. As a consequence, they are not sample-efficient, and practical applications with long trajectories are forced to consider restrictive policy spaces.

Learning to Optimize in Machine Learning Learning to Learn (L2L) is a form of meta-learning, aiming to use machine learning methods to learn better machine learning systems [53]. Research in the area has recently surged in popularity, resulting in various applications learning better neural network architectures [57, 41], hyper-parameters [44], initialization [16], and optimizers [4, 13, 11, 39, 56, 14]. As we are learning a component of an optimizer, our work is closely related to *Learning to Optimize* (L2O). Note that most L2O research [4, 39, 56, 14] focuses on learning better gradient-based methods (e.g. Adam [36]), as these are most commonly used to optimize neural networks. Notable exceptions are L2O applications to single-point [13] and multi-point [11] black-box optimization. One L2O approach [13, 4, 11] models iterative optimizers as a kind of recurrent neural network and trains it in a fully supervised fashion. More general RL methods have also been used in L2O, e.g. REPS [14], PPO [56], and GPS [39]. In this work, we apply GPS in a similar way. The main difference is that, instead of learning a simple first-order optimization method from scratch, we apply this method to dynamically configure a single parameter (step-size) in a state-of-the-art derivative-free method (CMA-ES).

3 Background on CMA-ES

Covariance Matrix Adaptation Evolution Strategy (CMA-ES) [24] is an evolutionary algorithm optimizing a continuous black-box function $f : \mathbb{R}^n \to \mathbb{R}$ by sampling individuals from a non-stationary multivariate normal search distribution $\mathcal{N}\left(m^{(g)}, {\sigma^{(g)}}^2 C^{(g)}\right)$, with mean $m^{(g)}$ (center), step-size $\sigma^{(g)}$ (scale) and covariance matrix $C^{(g)}$ (shape).

Initially, $C^{(0)} = I$ (identity matrix) and $m^{(0)}$, $\sigma^{(0)}$ are provided by the user. The algorithm then iteratively updates this search distribution to increase the probability of sampling successful individuals. Each generation g, CMA-ES first samples λ individuals $x_1^{(g+1)}, ..., x_\lambda^{(g+1)}$ and chooses the best μ points as the parents of generation *g+1*. Then CMA-ES shifts the mean by a weighted average of μ selected steps:

$$m^{(g+1)} = m^{(g)} + c_m \sum_{i=1}^{\mu} w_i \left(x_{i:\lambda}^{(g+1)} - m^{(g)}\right). \tag{1}$$

where $x_{i:\lambda}$ denotes the i-th best point in terms of the function value and c_m is a learning rate which is usually set to 1. Next, covariance matrix adaptation is performed, which amounts to learning a second order model of the underlying objective function. To control the step-size CMA-ES uses *Cumulative Step Length Adaptation (CSA)* [21]:

$$\sigma^{(g+1)} = \sigma^{(g)} exp\left(\frac{c_\sigma}{d_\sigma}\left(\frac{||p_\sigma^{(g+1)}||}{E||\mathcal{N}(0,I)||} - 1\right)\right), \tag{2}$$

where $c_\sigma < 1$ is the learning rate, $d_\sigma \approx 1$ is the damping parameter, and $p_\sigma^{(g+1)} \in \mathbb{R}^n$ is the conjugate evolution path at generation *g+1*:

$$p_\sigma^{(g+1)} = (1 - c_\sigma) p_\sigma^{(g)} + \sqrt{c_\sigma(2 - c_\sigma)\mu_{eff}} C^{(g)^{-\frac{1}{2}}} \frac{m^{(g+1)} - m^{(g)}}{\sigma^{(g)}}. \tag{3}$$

Note that alternatives for CSA have been proposed, e.g. making use of a success rule [30] or facilitate two-point step-size adaptation [20]. More generally, further research has resulted in many variants of CMA-ES suitable for a variety of different problems. A highly modular framework [48] exists that enables easy choice between 4 608 different versions of CMA-ES. This framework has further been used to demonstrate that, theoretically, switching only once during a run between configurations can yield performance improvements [47]. Simply using the switching rules proposed therein did not yield robust results in practice, but could be improved upon to yield better results [55].

4 Learning Step-Size Adaptation

In this section, we will first discuss how we can model the adaptation of the step-size of CMA-ES as a dynamic algorithm configuration problem and propose to use guided policy search to efficiently find well-performing step-size policies.

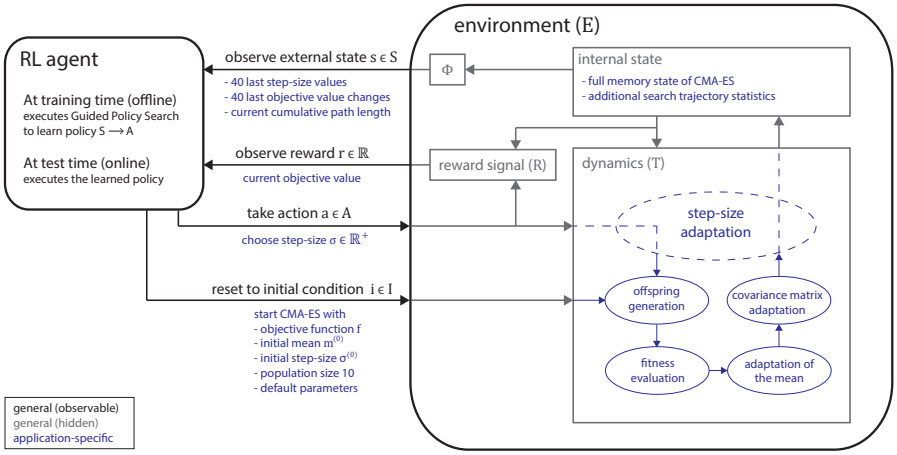


Fig. 1: Interaction of the RL agent with CMA-ES.

4.1 The General Objective

The general objective is to adjust step-size $\sigma^{(g+1)}$ for generation $g+1$ based on some state information s_g on how CMA-ES behaved so far. To achieve that, a probabilistic policy π is responsible for the adjustment:

$$\sigma^{(g+1)} \sim \pi(s_g) \tag{4}$$

Along the lines of DAC, we further say that a policy should not only perform well on a single function f, but generalize to many functions $f \in \mathcal{F}$. Note that the policy must not only depend on features of the search trajectory, but could also be enriched by context information about the function at hand. This allows the policy to easily distinguish between different functions and their characteristics.

Dynamic algorithm configuration allows us to minimize an arbitrary cost function $c : \Pi \times \mathcal{F} \to \mathbb{R}$ that defines how well our algorithm, here CMA-ES, performed by using a policy $\pi \in \Pi$ on a function $f \in \mathcal{F}$. Therefore, our objective is to find a policy π^* that optimally adjusts σ across a set of functions[4]:

$$\pi^* \in \arg\min_{\pi \in \Pi} \sum_{f \in \mathcal{F}} c(\pi, f) \tag{5}$$

4.2 Defining the Components

Having formally described the specific DAC problem at hand, we need to define all the components to apply reinforcement learning (RL) for solving it. In general, the RL paradigm [52] allows learning a policy π mapping observations $\Phi(s') \in S$ of an internal state $s' \in S'$ to actions A by optimizing some reward signal R

[4] We assume that the cost function is well-defined such that an optimal policy exists.

induced by transitions $T : S' \times A \to S'$. So, to solve our DAC problem for step-size adaptation in CMA-ES via RL, we need to define our problem as $\langle S, A, T, R \rangle$, where T is implicitly given by the dynamics of CMA-ES; see Figure 1.

The Step-Size Domain and the Action Space. In principle, the step-size parameter of CMA-ES is a positive, continuous scalar that needs to be adjusted by π. For this, we have two options: (i) discretizing it or (ii) directly optimizing in a continuous domain, which will represent the action space for our RL approach. We argue that the first option is not desirable, as a too fine grid might lead to a large action space with many potentially irrelevant choices; whereas a too coarse grid might not contain all relevant choices. Hence, we model A as the continuous domain of the step size parameter.

State Representation of CMA-ES. To enable DAC for the step-size, it is crucial that S encodes sufficient information about the CMA-ES run. Given that our aim is to learn from, and possibly improve over, the performance of CSA for step-size control, we encode the information CSA uses in the state. Additionally, we include information on the optimization process by keeping a history of a fixed number h of past step-size values (in our experiments, $h = 40$) and past objective values. Specifically, our chosen state description contains:

1. the current step-size value $\sigma^{(g)}$ (see Equation 2)
2. the current cumulative path length $p_\sigma^{(g)}$ (see Equation 2)
3. the history of changes in objective value (i.e. the differences between successive objective values from h previous iterations)
4. the step-size history from h previous iterations[5]

The Cost Function and the Reward. The overall objective of CMA-ES is to find the minimizer of a function f at hand. So, we can say that the cost function should directly reflect the function value found by CMA-ES. Because the optimization budget (e.g., the number of allowed function evaluations) is not always known beforehand, we argue that it is desired to optimize for any-time performance. Since RL maximizes a cumulative reward over time, we can simply define the reward function per step (i.e. generation) as the negative function value of the current incumbent. By doing that, we optimize for any-time performance.

4.3 Using Guided Policy Search for Efficient Learning of the Policy

Prior work showed [49, 51] that value-based RL can be used for learning a DAC policy. However, this approach is typically not very sample-efficient, making it a very expensive approach in general. For example, Biedenkapp et al. [10] needed more than 10 000 algorithm runs to learn a simple sigmoid function.

A key insight of our work is that RL does not need to learn a well-performing policy from scratch but can use existing self-adaptive heuristics as a teacher.

[5] When such a long history is not available yet, the missing values are filled with zeros.

Here, for example, we propose to use CSA as a teacher to learn a policy that either imitates or improves upon it. In addition to better learning stability of the policy, we will show in our experiments that learning a policy for step-size adaptation is comparably cheap by using less than 1 000 runs of CMA-ES.

Similar to Li and Malik [39] in learning to optimize, we propose to use guided policy search (GPS) under unknown dynamics [37] to learn step-size policies. In essence GPS learns arbitrary parameterized policies through supervised learning by fitting a policy to guiding trajectories [38, 37]. From teaching trajectories, a teacher distribution is computed such that it maximizes the reward and the agreement with the current policy. The policy parameters are then updated in a supervised fashion such that new sample trajectories produced by the policy do not deviate too far from the teacher. For a detailed explanation we refer to [37].

4.4 Extending GPS-based DAC by a Stronger Teacher

For our purposes, we initialize the teacher to fit trajectories generated by CMA-ES with CSA. This initial teacher thus closely resembles CSA. As GPS updates the teacher over time to improve the reward, the teacher is likely to move away from CSA over time as only student and teacher are constrained to stay close to each other. If both teacher and student stray too far from CSA, the learned policy might not be able to recover CSAs behaviour in cases where it is beneficial. Thus we would like to encourage the student policy to also continually learn from CSA, to gain a more diverse teaching experience.

Instead of restricting the student policies through hard divergence criterion to not go too far away from CSA, we propose to add additional sample trajectories from running CMA-ES with CSA and not only the teacher to train the student policy. Thereby CSA acts as an additional fixed teacher. We extend GPS by introducing a *sampling rate* to determine the fraction of sample trajectories obtained from CSA when training the policy. Finally, in order to ensure exploration during learning, the initial step-size values and values for the mean of the initial distribution for the functions in the training set are randomly sampled from a uniform distribution.

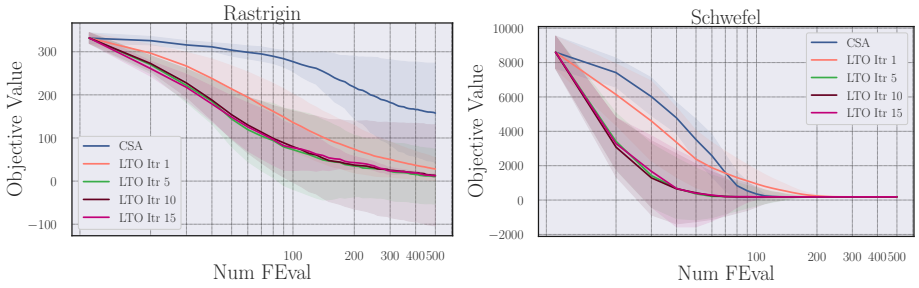
5 Experiments

In this section we empirically evaluate the effectiveness of our proposed approach. We demonstrate the ability of our policies to generalize to new settings.

5.1 Setup

For guided policy search we used the implementation provided by Li and Malik [39]. We incorporated our policy[6] in the python version of CMA-ES (pycma) in version 2.7.0 [22]. We only optimized the step-size adaptation with our approach and left all other pycma parameters as specified by the default, except we

[6] Code and trained policies available at https://github.com/automl/LTO-CMA



(a) (b)

Fig. 2: Performance comparison of CMA-ES default step-size adaptation (CSA) to that of our methods incumbent policy after 1, 5, 10 and 15 training iterations on the Rastrigin function (a) and the Schwefel function (b).

used a fixed population size of 10. As functions, we used a representative set with different characteristics as introduced by Hansen et al. in the BBOB-2009 [23].

We used 10 runs of SMAC [28, 40] to tune the initial step-size of CSA for each of the 10 considered BBOB functions individually, giving us a strong baseline. On the unseen functions we used an initial step-size of 0.5. In all experiments we used the same initial step-size for both our method and the baseline.

We trained our step-size policy for 50 steps (i.e. generations) of CMA-ES. We model the policy as a neural network consisting of two hidden layers with 50 hidden units each and ReLU activations. During training, the trajectory samples are obtained from the teaching policy with a probability of 0.7, whereas with a probability of 0.3 we sample trajectories from running CMA-ES with CSA. We obtain the final policy after training for 15 iterations of GPS.

We show performance comparisons of CMA-ES with the learned policy for step-size adaptation and CMA-ES with CSA from 25 independent runs of each method. The tables show an estimate of how likely it is for our learned policy to outperform CSA, based on pairwise comparisons of final objective values from the 25 runs for each method. The online appendix[7] describes this metric in detail, including its relation to statistical significance[8].

5.2 Function-Specific Policy

Comparison against our Teacher CSA We begin by exploring our method's ability to learn step-size policies when trained on a single 10D function for which we sampled 18 different starting points. In each training iteration of GPS, we evaluated CMA-ES 5 times on all starting conditions. In most cases, we already learn a well performing policy after 10 training iterations, which amounts only

[7] https://ml.informatik.uni-freiburg.de/papers/20-PPSN-LTO-CMA.pdf
[8] Estimates $\geq 0.64 \implies$ our learned policy significantly outperformed CSA ($\alpha = 0.05$)

	Sampling Rate									
	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
BentCigar	0.53	0.84	0.46	**0.96**	0.38	0.33	0.14	0.26	0.25	0.08
Discus	0.00	0.66	0.23	**0.74**	0.34	0.35	0.30	0.37	0.29	0.32
Ellipsoid	0.59	**0.97**	0.51	**0.97**	0.51	0.48	0.35	0.48	0.56	0.44
Katsuura	0.64	0.91	0.66	**0.96**	0.64	0.63	0.63	0.64	0.64	0.61
Rastrigin	0.81	0.94	0.83	**1.00**	0.97	0.87	0.79	0.85	0.79	0.80
Rosenbrock	0.67	0.28	0.43	**0.89**	0.61	0.17	0.12	0.51	0.57	0.22
Schaffers	0.75	0.68	0.87	0.78	0.92	**0.98**	0.45	0.57	0.90	0.94
Schwefel	0.93	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**
Sphere	0.77	0.92	0.48	0.78	0.58	0.25	0.94	**0.99**	0.93	0.94
Weierstrass	0.35	**1.00**	0.54	**1.00**	0.32	0.52	0.58	0.52	0.49	0.42
Average	0.60	0.82	0.60	0.91	0.63	0.56	0.53	0.62	0.64	0.58

Table 1: Probability of our method to outperform the baseline when training with different sampling rates. 1.0 indicates that we always outperform the baseline and 0.0 indicates we are always outperformed. The best sampling rate per function are marked in bold.

to $18 \times 5 \times 10 = 900$ runs of CMA-ES. Figure 2 depicts the training performance of our learned step-size policy after 1, 5, 10 and 15 training iterations of GPS on the Rastrigin and Rosenbrock functions. From Figure 2a we can see that even though our policy starts out with samples from the default step-size adaptation of CMA-ES, already after one iteration, the learned policy can outperform the hand-crafted baseline. After four more training steps, our learned policy continues improving and still outperforms CSA. Finally when having trained for 15 iterations, our learned policy readily outperforms CSA, leading not only to a much better final performance, but also to a much better anytime performance on the Rastrigin function. We observe a similar behaviour when training on the Schwefel function, but the learned policy does not drastically outperform CSA.

Studying the Sampling Rate We further used this setting to determine the influence of training length and sampling rate on the final performance of our policies, see Table 1. The sampling rate is crucial for our method as it determines how similar the learned policy's behavior is to CSA.

The performance of the learned policy improved by introducing sample trajectories from CSA compared to only sampling from the time-varying linear Gaussian teacher. Results on some functions are more strongly affected by this change, e.g. BentCigar, than others, such as Schwefel. The final row shows the average performance of the sampling rate over all 10 considered training functions. Further, it becomes apparent that a sampling rate of 0.3 results in the strongest performance of our method, indicating that sampling also from our default teacher can improve performance. As a conclusion of this meta-parameter study, we will use 0.3 for our following experiments on generalization.

	Trajectory Length						
	50	100	150	200	250	500	1000
BentCigar	**0.89**	0.00	0.00	0.00	0.00	0.05	0.04
Discus	0.90	**0.95**	0.76	0.40	0.00	0.00	0.00
Ellipsoid	**0.94**	0.92	0.90	0.86	0.61	0.00	0.00
Katsuura	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**
Rastrigin	**1.00**	0.81	0.80	0.83	0.92	0.73	0.74
Rosenbrock	**0.93**	0.77	0.78	0.90	0.62	0.24	0.04
Schaffers	**0.60**	0.55	0.40	0.39	0.48	0.39	0.57
Schwefel	**0.99**	0.52	0.76	0.79	0.87	0.84	0.65
Sphere	**0.89**	0.00	0.00	0.00	0.00	0.00	0.00
Weierstrass	0.97	0.97	0.89	0.92	**1.00**	**1.00**	**1.00**
Average	0.91	0.65	0.63	0.61	0.55	0.43	0.40

(a) Different Trajectory Lengths

	Dimensions					
	35	40	45	50	55	60
BentCigar	0.87	0.98	0.56	0.49	0.76	**1.00**
Discus	0.89	0.86	0.93	0.94	0.94	**0.97**
Ellipsoid	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**
Katsuura	0.92	0.92	0.96	**1.00**	0.96	0.87
Rastrigin	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**
Rosenbrock	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**
Schaffers	0.31	0.58	0.78	**0.87**	0.76	0.74
Schwefel	**1.00**	0.96	0.96	**1.00**	**1.00**	0.98
Sphere	0.41	0.38	0.56	0.65	0.64	**0.72**
Weierstrass	0.97	**1.00**	0.95	**1.00**	**1.00**	0.93
Average	0.84	0.87	0.87	0.89	0.91	0.92

(b) Different # Dimensions

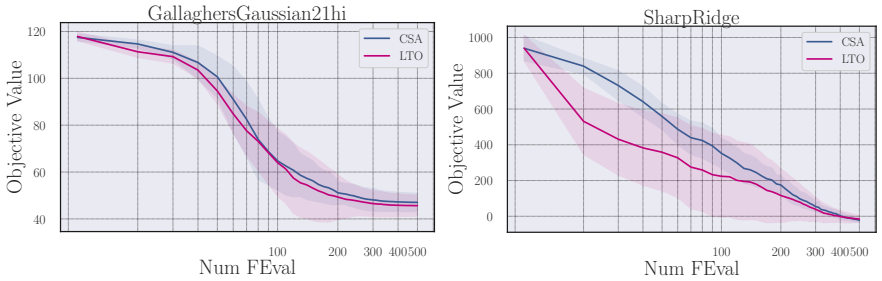
Table 2: Probability of our method to outperform the baseline (a) for varying trajectory lengths, when having only trained with trajectories of length 50, and (b) for different dimensions when training them on functions of dimension $5-30$ and applying the learned policies to functions of dimensionality > 30.

Generalization to Longer Trajectory Length Finally, in this setting we explore the capability of the agent to transfer to longer trajectories. During training we opted to limit the training samples to be of maximal length 50, which corresponds to 500 function evaluations, to keep the cost for training low. Naturally the question thus arises if it is possible to further make use of such policies on longer optimization trajectories. From Table 2a we can observe that, even if a policy is trained with trajectories of at most 500 function evaluations, the policies are generally capable of generalizing to optimization trajectories that are 5 times longer while struggling to generalize to even longer trajectories.[9] On functions where the learned policies perform very well, only a small performance decrease is noticeable over a longer trajectory. On other functions the final performance lacks behind that of the handcrafted baseline over longer optimization trajectories, whereas on Weierstrass, the opposite is the case. On average, we can observe a decline in final performance of our learned policy, the further the optimization trajectory length is from the one used originally for training. A limiting factor as of yet is scaling the training to much longer trajectories. With increased trajectory length more training iterations will be needed to learn well performing policies.

5.3 Function-Class Specific Policy

We are generally not interested in policies that are only of use for one specific function; a more desirable policy would be capable of handling a broader range

[9] The learned policies outperform CSA on anytime performance as shown in the Appendix, but CSA is better in terms of end objective values.



(a) (b)

Fig. 3: Optimization trajectories of CMA-ES using CSA (blue) and our learned policy on two prior unseen test functions. The solid lines depict the mean performance and the shaded area the standard deviation over 25 repetitions.

of functions. As a first step, we are interested in generalizing to similar functions of a specific function class. A very interesting, yet challenging task is hereby to generalize to higher dimensions. For this purpose we trained our policies on functions of dimension $5 - 30$ and evaluated them on dimensions $35 - 60$.

From Table 2b we can see that with increasing dimensionality, the probability that our policies outperform the handcrafted baseline actually *increases*. Upon inspection of the results, we see that with increasing dimensionality, the baseline optimization trajectories need more and more generations before reaching a good performance. Similarly, with increase in dimensionality, optimization trajectories guided by our policies require more generations to reach good final performances, however they are less affected by the dimensionality than the baseline. Especially on functions like Rosenbrock or Ellipsoid this effect seems to be very strongly pronounced. We can observe this trend for both training and testing our policies (see appendix for results on training).

5.4 Generalization to New Functions

Policies scaling to higher dimensions already promise great generalization capability. However, in practice, the problems, to which a solver is applied, could be fairly heterogeneous. To look into a more realistic scenario, we trained our agent on the 10 black-box functions we have mentioned before and assess its generalization capability on 12 black-box functions unseen during training.

Figure 3 shows two exemplary optimization trajectories that are achievable with our learned policies, compared to that of the default CSA. On Gallhager's Gaussian 21-hi we see that the optimization trajectory of CMA-ES with our learned policy closely resembles that of the handcrafted baseline as the step-sizes follow the same trend, see Figure 3a. On SharpRidge (Figure 3b) the learned policy is able to find well performing regions quicker; however in the end the baseline catches up.

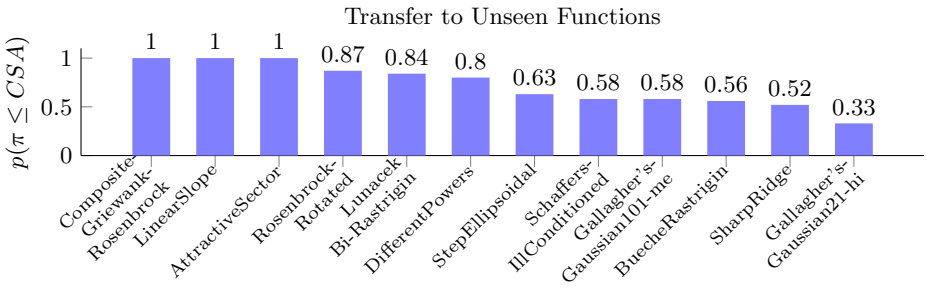


Fig. 4: Probability of our learned policies outperforming the default baseline on prior unseen test functions when training on all 10 BBOB functions.

Figure 4 summarizes the result for all 12 test functions. On 6 out of the 12 test functions, the learned policy significantly (≥ 0.64, $\alpha = 0.05$) outperformed the baseline, while being significantly outperformed (≤ 0.36) on one.

6 Conclusion

We demonstrated that we can automatically learn policies to dynamically configure the mutation step-size parameter of CMA-ES using reinforcement learning. To the best of our knowledge, we are the first to use policy search for the dynamic configuration of evolutionary algorithms, rather than value-based reinforcement learning. In particular, we described how *guided* policy search can be used to learn configuration policies, starting from a known handcrafted default policy. We conducted a comprehensive empirical investigation, and observed that (i) the learned policies are capable of outperforming the default policy on a wide range of black-box optimization problems; (ii) using a fixed teacher can further improve the performance; (iii) our learned policies can generalize to higher dimensions as well as to unseen functions.

These results open the door for promising future research in which policy search is used to learn policies that jointly configure multiple parameters (e.g. population *and* step-size) of CMA-ES. Another line of future research could improve the employed policy search mechanism, e.g. by learning from a variety of teachers at the same time. A more diverse set of teachers, might facilitate even better generalization as the learned policies could make use of strengths of individual teachers on varying problem domains. Finally, the development of a benchmark platform for dynamic algorithm configuration would facilitate apple-to-apple comparisons of different reinforcement learning techniques, driving future research.

Acknowledgements. The authors acknowledge funding by the Robert Bosch GmbH.

Bibliography

[1] Adriaensen, S., Nowé, A.: Towards a white box approach to automated algorithm design. In: Kambhampati, S. (ed.) Proceedings of the 26th International Joint Conference on Artificial Intelligence (IJCAI'16). pp. 554–560 (2016)

[2] Aleti, A., Moser, I.: A systematic literature review of adaptive parameter control methods for evolutionary algorithms. ACM Comput. Surv. **49**(3), 56:1–56:35 (2016)

[3] Andersson, M., Bandaru, S., Ng, A.H.: Tuning of multiple parameter sets in evolutionary algorithms. In: Proceedings of the Genetic and Evolutionary Computation Conference 2016. pp. 533–540 (2016)

[4] Andrychowicz, M., Denil, M., Colmenarejo, S.G., Hoffman, M.W., Pfau, D., Schaul, T., de Freitas, N.: Learning to learn by gradient descent by gradient descent. In: Lee, D., Sugiyama, M., von Luxburg, U., Guyon, I., Garnett, R. (eds.) Proceedings of the 30th International Conference on Advances in Neural Information Processing Systems (NeurIPS'16). pp. 3981–3989 (2016)

[5] Ansótegui, C., Malitsky, Y., Sellmann, M.: Maxsat by improved instance-specific algorithm configuration. In: Brodley, C., Stone, P. (eds.) Proceedings of the Twenty-eighth National Conference on Artificial Intelligence (AAAI'14). pp. 2594–2600. AAAI Press (2014)

[6] Ansótegui, C., Pon, J., Sellmann, M., Tierney, K.: Reactive dialectic search portfolios for maxsat. In: S.Singh, Markovitch, S. (eds.) Proceedings of the Conference on Artificial Intelligence (AAAI'17). AAAI Press (2017)

[7] Ansótegui, C., Sellmann, M., Tierney, K.: A gender-based genetic algorithm for the automatic configuration of algorithms. In: Gent, I. (ed.) Proceedings of the Fifteenth International Conference on Principles and Practice of Constraint Programming (CP'09). Lecture Notes in Computer Science, vol. 5732, pp. 142–157. Springer (2009)

[8] Battiti, R., Campigotto, P.: An investigation of reinforcement learning for reactive search optimization. In: Hamadi, Y., Monfroy, E., Saubion, F. (eds.) Autonomous Search, pp. 131–160. Springer (2012)

[9] Battiti, R., Brunato, M., Mascia, F.: Reactive search and intelligent optimization, vol. 45. Springer Science & Business Media (2008)

[10] Biedenkapp, A., Bozkurt, H.F., Eimer, T., Hutter, F., Lindauer, M.: Dynamic Algorithm Configuration: Foundation of a New Meta-Algorithmic Framework. In: Lang, J., Giacomo, G.D., Dilkina, B., Milano, M. (eds.) Proceedings of the Twenty-fourth European Conference on Artificial Intelligence (ECAI'20) (Jun 2020)

[11] Cao, Y., Chen, T., Wang, Z., Shen, Y.: Learning to optimize in swarms. In: Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., Garnett, R. (eds.) Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, (NeurIPS'19). pp. 15018–15028 (2019)

[12] Chen, F., Gao, Y., Chen, Z., Chen, S.: Scga: Controlling genetic algorithms with sarsa (0). In: International Conference on Computational Intelligence for Modelling, Control and Automation and International Conference on Intelligent Agents, Web Technologies and Internet Commerce (CIMCA-IAWTIC'06). vol. 1, pp. 1177–1183. IEEE (2005)

[13] Chen, Y., Hoffman, M., Colmenarejo, S., Denil, M., Lillicrap, T., Botvinick, M., de Freitas, N.: Learning to learn without gradient descent by gradient descent. In: Precup, D., Teh, Y. (eds.) Proceedings of the 34th International Conference on Machine Learning (ICML'17). vol. 70, pp. 748–756. Proceedings of Machine Learning Research (2017)

[14] Daniel, C., Taylor, J., Nowozin, S.: Learning step size controllers for robust neural network training. In: Schuurmans, D., Wellman, M. (eds.) Proceedings of the Thirtieth National Conference on Artificial Intelligence (AAAI'16). AAAI Press (2016)

[15] Eiben, A., Horváth, M., Kowalczyk, W., Schut, M.: Reinforcement learning for online control of evolutionary algorithms. In: Brueckner, S., Hassas, S., Jelasity, M., Yamins, D. (eds.) Proceedings of Engineering Self-Organising Systems (ESOA). Lecture Notes in Computer Science, vol. 4335, pp. 151–160. Springer (2007)

[16] Finn, C., Abbeel, P., Levine, S.: Model-agnostic meta-learning for fast adaptation of deep networks. In: Precup, D., Teh, Y. (eds.) Proceedings of the 34th International Conference on Machine Learning (ICML'17). vol. 70, pp. 1126–1135. Proceedings of Machine Learning Research (2017)

[17] Fuks, L., Awad, N., Hutter, F., Lindauer, M.: An evolution strategy with progressive episode lengths for playing games. In: Kraus, S. (ed.) Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence (IJCAI). pp. 1234–1240. ijcai.org (2019)

[18] Gaspero, L.D., Urli, T.: Evaluation of a family of reinforcement learning cross-domain optimization heuristics. In: Hamadi, Y., Schoenauer, M. (eds.) Proceedings of the Sixth International Conference on Learning and Intelligent Optimization (LION'12). Lecture Notes in Computer Science, vol. 7219, pp. 384–389. Springer (2012)

[19] Hansen, N.: The CMA evolution strategy: a comparing review. In: Lozano, J., Larranaga, P., Inza, I., Bengoetxea, E. (eds.) Towards a new evolutionary computation. Advances on estimation of distribution algorithms, pp. 75–102. Springer (2006)

[20] Hansen, N.: CMA-ES with two-point step-size adaptation. arXiv:0805.0231 [cs.NE] (2008)

[21] Hansen, N.: The CMA evolution strategy: A tutorial. arXiv:1604.00772v1 [cs.LG] (2016)

[22] Hansen, N., Akimoto, Y., Baudis, P.: CMA-ES/pycma on GitHub. Zenodo, DOI:10.5281/zenodo.2559634 (Feb 2019). https://doi.org/10.5281/zenodo.2559634, https://doi.org/10.5281/zenodo.2559634

[23] Hansen, N., Finck, S., Ros, R., Auger, A.: Real-Parameter Black-Box Optimization Benchmarking 2009: Noiseless Functions Definitions. Research Report RR-6829, INRIA (2009)
[24] Hansen, N., Ostermeier, A.: Convergence properties of evolution strategies with the derandomized covariance matrix adaptation: The $(\mu/\mu_I, \lambda)$-CMA-ES. Proceedings of the 5th European Congress on Intelligent Techniques and Soft Computing p. 650–654 (1997)
[25] Hansen, N., Ostermeier, A.: Completely derandomized self-adaptation in evolution strategies. Evolutionary Computation **9**, 159–195 (2001)
[26] Hoos, H.: Programming by optimization. Communications of the ACM **55**(2), 70–80 (2012)
[27] Hutter, F., Hoos, H., Leyton-Brown, K.: Automated configuration of mixed integer programming solvers. In: Lodi, A., Milano, M., Toth, P. (eds.) Proceedings of the Seventh International Conference on Integration of AI and OR Techniques in Constraint Programming (CPAIOR'10). Lecture Notes in Computer Science, vol. 6140, pp. 186–202. Springer (2010)
[28] Hutter, F., Hoos, H., Leyton-Brown, K.: Sequential model-based optimization for general algorithm configuration. In: Coello, C. (ed.) Proceedings of the Fifth International Conference on Learning and Intelligent Optimization (LION'11). Lecture Notes in Computer Science, vol. 6683, pp. 507–523. Springer (2011)
[29] Hutter, F., Lindauer, M., Balint, A., Bayless, S., Hoos, H., Leyton-Brown, K.: The configurable SAT solver challenge (CSSC). Artificial Intelligence **243**, 1–25 (2017)
[30] Igel, C., Hansen, N., Roth, S.: Covariance matrix adaptation for multi-objective optimization. Evolutionary Computation **15**, 1–28 (2001)
[31] Igel, C.: Neuroevolution for reinforcement learning using evolution strategies. In: The 2003 Congress on Evolutionary Computation, 2003. CEC'03. vol. 4, pp. 2588–2595. IEEE (2003)
[32] Kadioglu, S., Sellmann, M., Wagner, M.: Learning a reactive restart strategy to improve stochastic search. In: International Conference on Learning and Intelligent Optimization. pp. 109–123. Springer (2017)
[33] Karafotias, G., Eiben, A., Hoogendoorn, M.: Generic parameter control with reinforcement learning. In: Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation. pp. 1319–1326 (2014)
[34] Karafotias, G., Hoogendoorn, M., Eiben, Á.: Parameter control in evolutionary algorithms: Trends and challenges. IEEE Trans. Evolutionary Computation **19**(2), 167–187 (2015)
[35] Karafotias, G., Smit, S., Eiben, A.: A generic approach to parameter control. In: European Conference on the Applications of Evolutionary Computation. pp. 366–375. Springer (2012)
[36] Kingma, D., Ba, J.: Adam: A method for stochastic optimization. In: Proceedings of the International Conference on Learning Representations (ICLR'15) (2015), published online: iclr.cc
[37] Levine, S., Abbeel, P.: Learning neural network policies with guided policy search under unknown dynamics. In: Ghahramani, Z., Welling, M., Cortes,

C., Lawrence, N., Weinberger, K. (eds.) Proceedings of the 28th International Conference on Advances in Neural Information Processing Systems (NeurIPS'14). pp. 1071–1079 (2014)
[38] Levine, S., Koltun, V.: Guided policy search. In: Dasgupta, S., McAllester, D. (eds.) Proceedings of the 30th International Conference on Machine Learning (ICML'13). pp. 1–9. Omnipress (2013)
[39] Li, K., Malik, J.: Learning to optimize. In: Proceedings of the International Conference on Learning Representations (ICLR'17) (2017), published online: iclr.cc
[40] Lindauer, M., Eggensperger, K., Feurer, M., Falkner, S., Biedenkapp, A., Hutter, F.: SMAC v3: Algorithm configuration in Python. https://github.com/automl/SMAC3 (2017)
[41] Liu, H., Simonyan, K., Yang, Y.: DARTS: differentiable architecture search. In: Proceedings of the International Conference on Learning Representations (ICLR'19) (2019), published online: iclr.cc
[42] López-Ibáñez, M., Dubois-Lacoste, J., Stützle, T., Birattari, M.: The irace package, iterated race for automatic algorithm configuration. Tech. rep., IRIDIA, Université Libre de Bruxelles, Belgium (2011), http://iridia.ulb.ac.be/IridiaTrSeries/IridiaTr2011-004.pdf
[43] López-Ibáñez, M., Stützle, T.: Automatic configuration of multi-objective ACO algorithms. In: Dorigo, M., M-Birattari, Caro, G.D., Doursat, R., Engelbrecht, A.P., Floreano, D., Gambardella, L., Groß, R., Sahin, E., Sayama, H., Stützle, T. (eds.) Proceedings of the Seventh International Conference on Swarm Intelligence (ANTS'10). pp. 95–106. Lecture Notes in Computer Science, Springer (2010)
[44] Maclaurin, D., Duvenaud, D., Adams, R.: Gradient-based hyperparameter optimization through reversible learning. In: Bach, F., Blei, D. (eds.) Proceedings of the 32nd International Conference on Machine Learning (ICML'15). vol. 37, pp. 2113–2122. Omnipress (2015)
[45] Muller, S., Schraudolph, N., Koumoutsakos, P.: Step size adaptation in evolution strategies using reinforcement learning. In: Proceedings of the 2002 Congress on Evolutionary Computation. CEC'02 (Cat. No. 02TH8600). vol. 1, pp. 151–156. IEEE (2002)
[46] Pettinger, J., Everson, R.: Controlling genetic algorithms with reinforcement learning. In: Proceedings of the 4th Annual Conference on Genetic and Evolutionary Computation. pp. 692–692 (2002)
[47] van Rijn, S., Doerr, C., Bäck, T.: Towards an adaptive CMA-ES configurator. In: Auger, A., Fonseca, C.M., Lourenço, N., Machado, P., Paquete, L., Whitley, L.D. (eds.) Proceedings of the 15th International Conference on Parallel Problem Solving from Nature (PPSN'18). Lecture Notes in Computer Science, vol. 11101, pp. 54–65. Springer (2018)
[48] van Rijn, S., Wang, H., van Leeuwen, M., Bäck, T.: Evolving the structure of evolution strategies. In: 2016 IEEE Symposium Series on Computational Intelligence (SSCI). pp. 1–8. IEEE (2016)
[49] Sakurai, Y., Takada, K., Kawabe, T., Tsuruta, S.: A method to control parameters of evolutionary algorithms by using reinforcement learning. In:

Proceedings of the Sixth International Conference on Signal-Image Technology and Internet Based Systems. pp. 74–79. IEEE (2010)
[50] Salimans, T., Ho, J., Chen, X., Sutskever, I.: Evolution strategies as a scalable alternative to reinforcement learning. arXiv:1703.03864 [stat.ML] (2017)
[51] Sharma, M., Komninos, A., López-Ibáñez, M., Kazakov, D.: Deep reinforcement learning based parameter control in differential evolution. In: Proceedings of the Genetic and Evolutionary Computation Conference. pp. 709–717 (2019)
[52] Sutton, R.S., Barto, A.G.: Reinforcement learning: An introduction. MIT press (2018)
[53] Thrun, S., Pratt, L.: Learning to learn. Springer Science & Business Media (2012)
[54] Verdooren, L.: Extended tables of critical values for wilcoxon's test statistic. Biometrika **50**(1-2), 177–186 (1963)
[55] Vermetten, D., van Rijn, S., Bäck, T., Doerr, C.: Online selection of CMA-ES variants. In: Auger, A., Stützle, T. (eds.) Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'19). pp. 951–959. ACM (2019)
[56] Xu, Z., Dai, A.M., Kemp, J., Metz, L.: Learning an adaptive learning rate schedule. arXiv:1909.09712 [cs.LG] (2019)
[57] Zoph, B., Le, Q.V.: Neural architecture search with reinforcement learning. In: Proceedings of the International Conference on Learning Representations (ICLR'17) (2017), published online: iclr.cc

Supplementary Material for: Learning Step-Size Adaptation in CMA-ES

Gresa Shala[1], André Biedenkapp[1], Noor Awad[1], Steven Adriaensen[1], Marius Lindauer[2], and Frank Hutter[1,3]

[1] University of Freiburg, Germany
[2] Leibniz University Hannover, Germany
[3] Bosch Center for Artificial Intelligence

A Available Software and Trained Policies

To enable other researchers to use our code, as well as trained models both are publicly available at https://github.com/automl/LTO-CMA. All scripts used to generate and plot our results, as well as the logged data are provided in the repository. Further, we provide examples of how to use our trained policy networks with the python version of CMA-ES (pycma) in version 2.7.0.

B Influence of Different Reward Scales

A drawback of prior methods of dynamic algorithm configuration through model-free RL is the need to learn the value function to find a well performing policy. With such value-based approaches learning across environments with very different reward scales is more challenging than with guided policy search, since the value function and therefore the policy can quickly be dominated by an environment with a large reward scale. Guided policy search on the other hand does not learn a value function to determine a well performing policy. GPS rather makes use of the reward signal to determine in which direction a better final reward can be achieved. On each condition it optimizes trajectory controllers individually with respect to reward and then learns policies that are similar to those teaching trajectories. Thus the learning policy is not influenced by different reward scales and simply learns to imitate teachers that were optimized for each function individually.

C Experimental Setup

We evaluated our approach on a compute cluster with nodes equipped with two Intel Xeon Gold 6242 32-core CPUs, 20 MB cache and and 188GB (shared) RAM running Ubuntu 18.04 LTS 64 bit.

Function Specific Policy For each of the 10 BBOB functions, we trained the policies on a set of 18 conditions consisting of the same function, but with different initialization values for the mean and step-size of CMA-ES.

Function Class Specific Policy For each of the 10 BBOB functions, we trained the policies on a set of 48 conditions consisting of the same function, but with different dimensionality(5D, 10D, 15D, 20D, 25D, 30D), initialization values for the mean and step-size of CMA-ES.

General Policy for BBOB We trained the policy on a set of 80 conditions consisting of 8 conditions for each function with the same dimensionality (10D), but with different initialization values for the mean and step-size of CMA-ES.

Used Metric We evaluated each method for n runs. To compare the resulting performance of using our learned policy to that of the handcrafted baseline, we compared the final performance of each of the 25 runs of our method to that of the handcrafted baseline. We count how often our method outperforms the baseline over all comparisons which gives us a probability of our policy π outperforming the baseline CSA as

$$p(\pi < CSA) = \frac{\sum_i^n \sum_j^n \mathbb{1}_{\pi_i < CSA_j}}{n^2} \tag{1}$$

where $\mathbb{1}_{\pi_i < CSA_j}$ is the indicator function showing if our policy resulted in a lower final objective value than the baseline when comparing runs i and j.

Statistical Significance: The performance metric used (see Equation 1) can easily be interpreted statistically as there exists a correspondence between $p(\pi < CSA)$ and the 'sum of ranks' statistic (W) used in the Wilcoxon rank-sum test, i.e. $p(\pi < CSA) = \frac{W_{\max} - W}{W_{\max} - W_{\min}}$ with $W_{\max} = \frac{n(3n+1)}{2}$ and $W_{\min} = \frac{n(n+1)}{2}$. As such, we can use critical values of the test statistic W [54] to derive critical values for $p(\pi < CSA)$, allowing us to draw conclusions about the significance of our results simply by comparing the value of this metric to a fixed threshold. Table C1 gives these thresholds for different common confidence levels α with $n = 25$.

p-value $< \alpha =$	**0.1**	**0.05**	**0.025**	**0.01**	**0.005**	**0.001**
$\mathbf{W} <$	570	552	536	517	505	480
$\mathbf{p}(\pi < \mathbf{CSA}) \geq$	0.61	0.64	0.67	0.70	0.72	0.76

Table C1: Critical values in terms of W ("sum of ranks" statistic) and $p(\pi < CSA)$ ("probability of outperforming" metric) for a single-sided Wilcoxon's rank-sum test with null-hypothesis: "CMA-ES with CSA performs at least as good as with our learned controller π" at different confidence levels α and $n = 25$.

	Sampling Rate									
	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
BentCigar	0.00	0.00	0.19	**1.00**	0.00	0.00	0.00	0.00	0.00	0.00
Discus	0.00	0.00	0.00	**1.00**	0.47	0.06	0.00	0.00	0.00	0.00
Ellipsoid	0.00	0.00	0.00	**1.00**	0.00	0.00	0.00	**1.00**	0.00	0.00
Katsuura	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**
Rastrigin	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**
Rosenbrock	**1.00**	0.60	**1.00**	**1.00**	**1.00**	0.00	0.19	**1.00**	**1.00**	0.00
Schaffers	0.00	0.00	0.00	**1.00**	0.00	0.00	0.00	0.00	0.00	0.00
Schwefel	**1.00**	0.00	0.00	**1.00**	0.00	**1.00**	0.00	**1.00**	0.00	**1.00**
Sphere	0.00	**1.00**	0.00	**1.00**	0.00	0.00	**1.00**	0.00	0.00	0.00
Weierstrass	0.00	**1.00**	**1.00**	**1.00**	0.00	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**
Average	0.40	0.46	0.42	**1.00**	0.35	0.41	0.42	0.60	0.40	0.40

Table D1: Probability of our approach outperforming CSA, in terms of AUC, for different dimensions on 10 BBOB functions with different sampling rates.

D Additional Analysis using Area Under the Curve

Here we show results of anytime performance comparisons between our approach and CSA. The reported results in the tables are computed using the same metric as described in Section C but we compare the AUC values instead of the final performance values. Table numbers here correspond to table numbers in the main paper. Table D1 shows the influence of different sampling rates on the performance, in terms of AUC, of our approach. This table confirms that also in terms of anytime performance a low sampling rate of 0.3 seems to perform best whereas smaller or larger sampling rates are detrimental.

Transfer to Longer Trajectories Table D2 shows the performance of our method when transferring to higher dimensions as well as to longer optimization trajectories. When transferring to longer optimization trajectories, our policy was trained for 500 function evaluations, i.e. 50 generations. We can see from Table D2a that our method is capable of outperforming the baseline CSA in terms of anytime performance. When we compare it to Table D2a of the main paper however we can see that the final performance gets worse, the further the optimization trajectory length is from the training setting. Plots in the following sections of this appendix show that, especially in the early stages of optimization, our learned policy very much outperforms the baseline. This early lead in optimization performance gives our method a much better AUC than the baseline.

Transfer to Higher Dimensions Table D2b shows the ability to transfer to higher dimensions, having trained the agent on functions of lower dimensions (i.e. 5D - 30D). We can see that up to 55D our learned policies AUC seems to be stay nearly the same. However for 60D the anytime performance becomes worse.

	Trajectory Length						
	50	100	150	200	250	500	1000
BentCigar	0.00	**1.00**	0.00	**1.00**	**1.00**	0.00	0.00
Discus	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**
Ellipsoid	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**
Katsuura	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**
Rastrigin	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**
Rosenbrock	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**
Schaffers	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**
Schwefel	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**
Sphere	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**
Weierstrass	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**
Average	0.90	**1.00**	0.90	**1.00**	**1.00**	0.90	0.90

(a) Different Trajectory Lengths

	Dimensions					
	35	40	45	50	55	60
BentCigar	0.00	0.00	0.00	0.00	**0.28**	0.00
Discus	**1.00**	0.00	**1.00**	**1.00**	0.00	0.00
Ellipsoid	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**
Katsuura	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	0.00
Rastrigin	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**
Rosenbrock	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**
Schaffers	0.00	0.00	0.00	0.00	**1.00**	0.00
Schwefel	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**
Sphere	0.00	0.00	0.00	0.00	**1.00**	0.00
Weierstrass	0.00	**1.00**	0.00	**1.00**	**1.00**	**1.00**
Average	0.70	0.70	0.60	0.70	0.83	0.50

(b) Different # Dimensions

Table D2: Probability of our method to outperform the baseline (a) for varying trajectory lengths, when having only trained with trajectories of length 50, and (b) for different dimensions when training them on functions of dimension $5-30$ and applying the learned policies to functions of dimensionality > 30.

This stands in contrast to our analysis in the main paper, using only the final performance, where with increasing dimensionality we could observe better final values.

E Performance Comparison across Training Iterations

In this section we provide additional plots for Section 5.2 of the main paper. We visually compare the performance of our proposed method to the baseline at different iterations of the training process. Figures 1a to 1j depict the performance of our method after 1, 5, 10 and 15 training iterations as well as the baseline performance when training only on the respective functions.

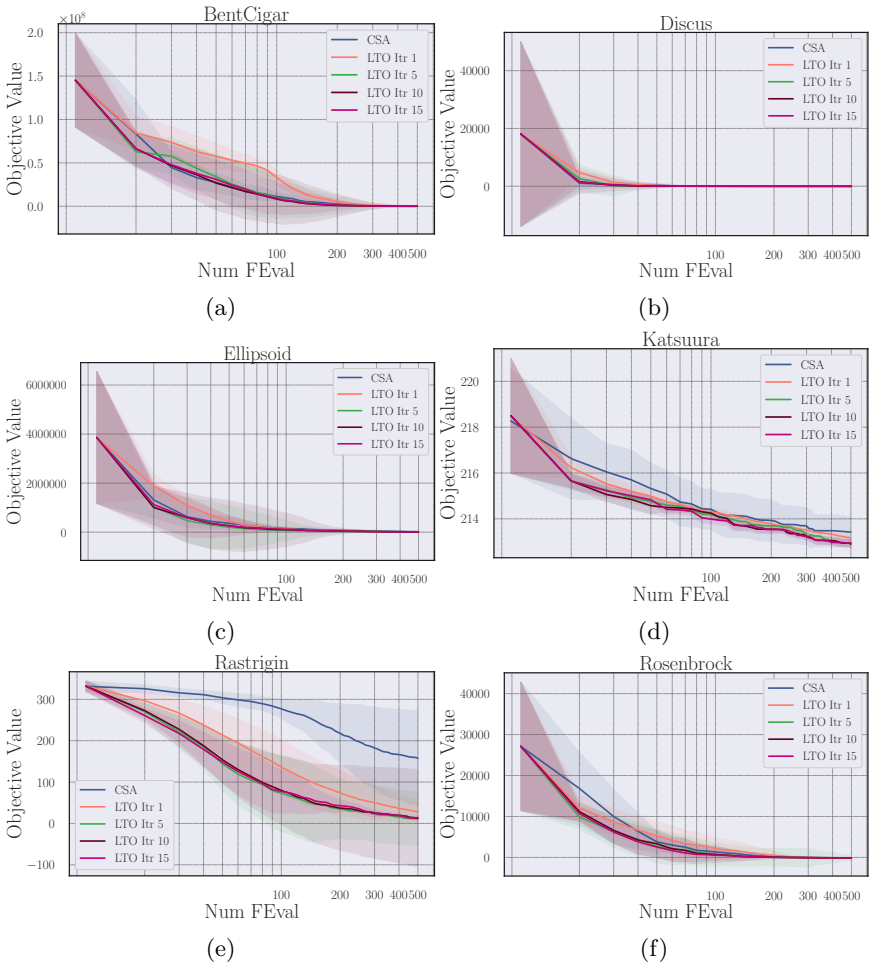


Fig. 1: Performance comparison of CMA-ES default step-size adaptation (CSA) to that of our methods incumbent policy after 1, 5, 10 and 15 training iterations of GPS on 10 BBOB functions.

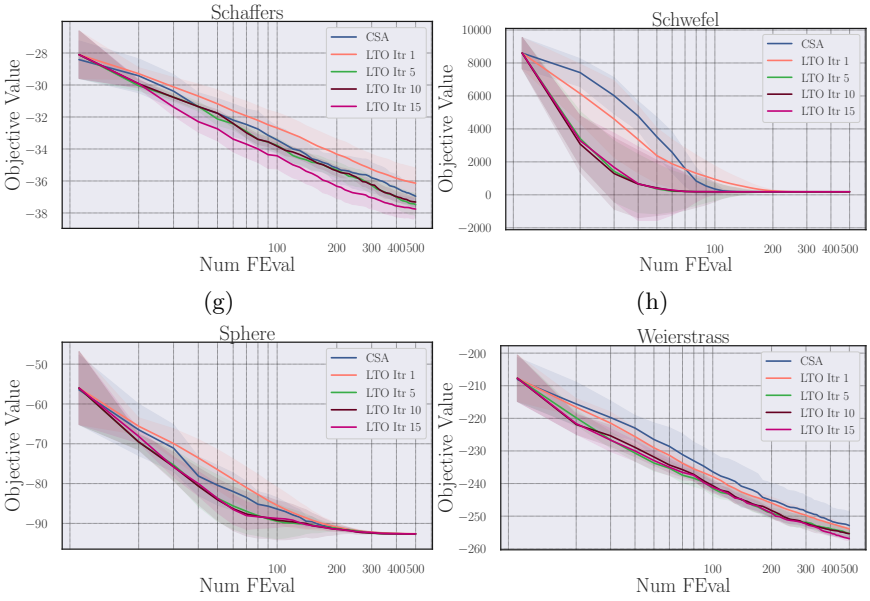


Fig. 1: Performance comparison of CMA-ES default step-size adaptation (CSA) to that of our methods incumbent policy after 1, 5, 10 and 15 training iterations of GPS on 10 BBOB functions.

F Sampling Rate

In this section we provide additional plots for Section 5.2 / Table 1 of the main paper. We visually compare the performance of our proposed method using a sampling rate of 0.3, vanilla GPS (i.e. sampling rate of 0) and the baseline (i.e. sampling rate of 1.0). Figure 2 shows the different optimization trajectories of CMA using the different step-size policies. Figure 3 depicts the corresponding step-size policies.

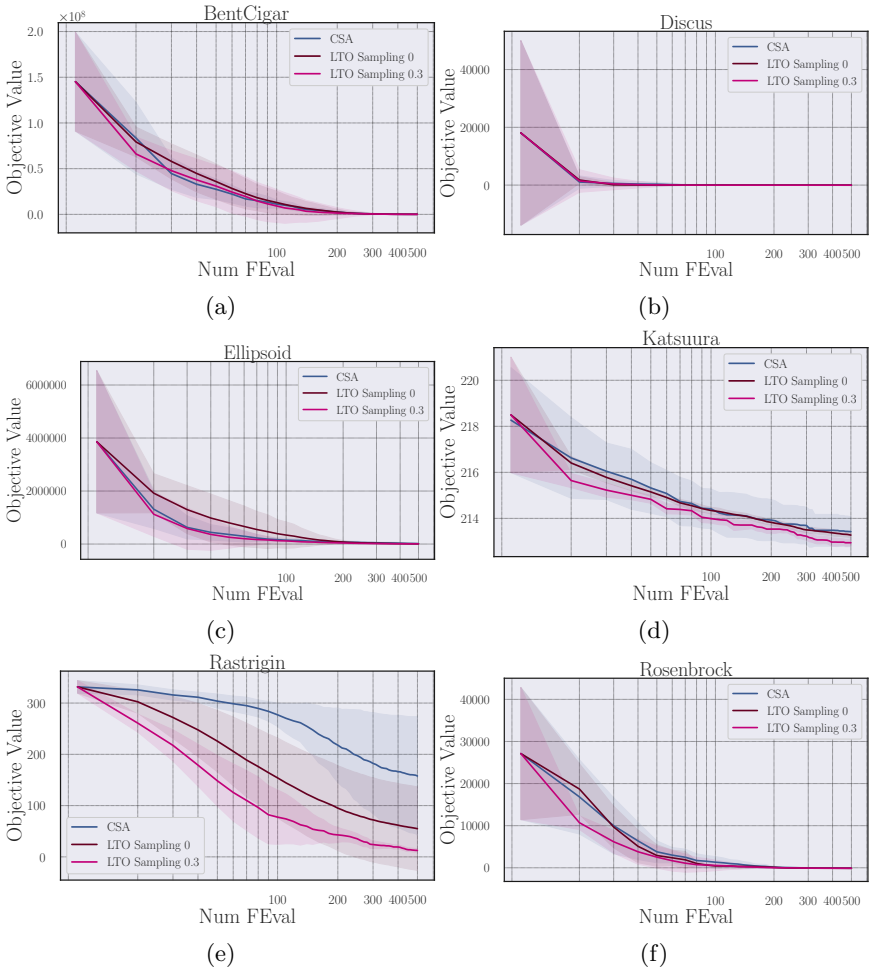


Fig. 2: Performance comparison of CMA-ES default step-size adaptation (CSA) to that of our method with a sampling rate of 0 and 0.3 on 10 BBOB functions.

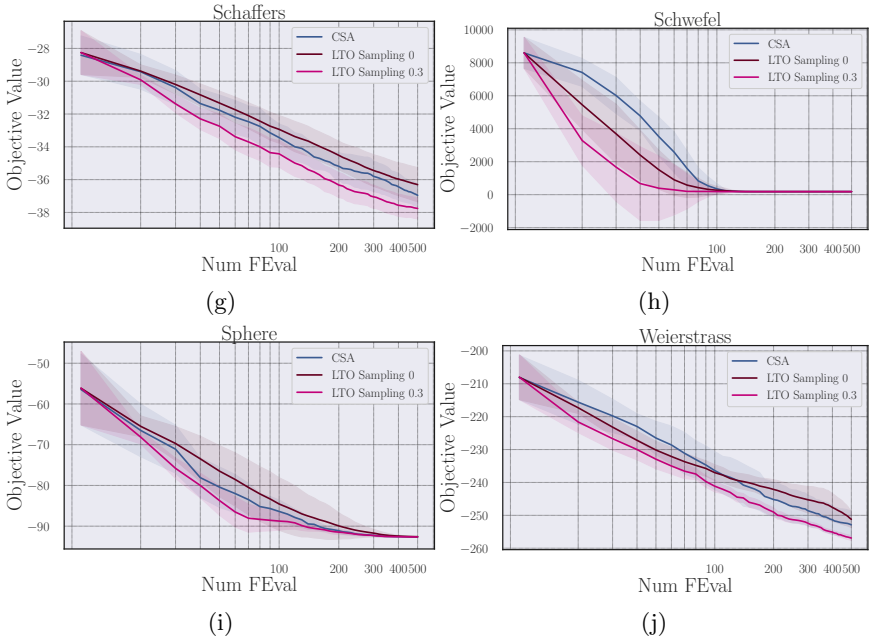


Fig. 2: Performance comparison of CMA-ES default step-size adaptation (CSA) to that of our method with a sampling rate of 0 and 0.3 on 10 BBOB functions.

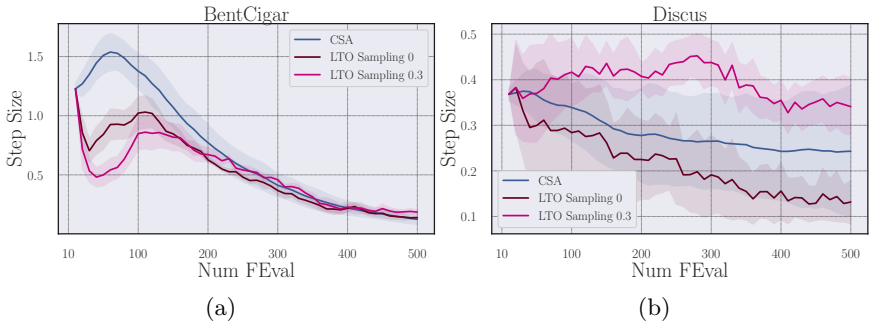


Fig. 3: Step-size adaptation comparison of CSA to that of our learned policies with a sampling rate of 0 and 0.3 on 10 BBOB functions.

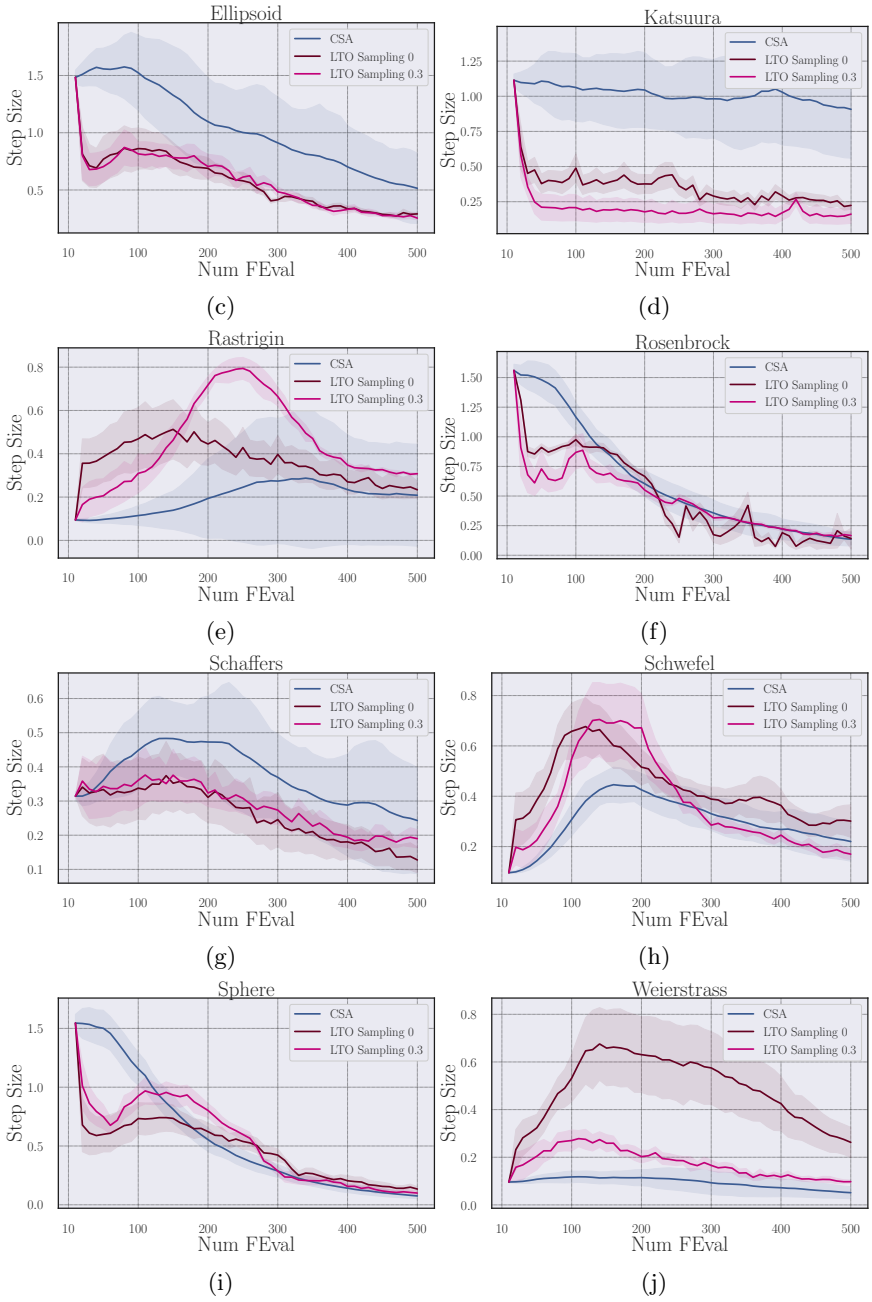
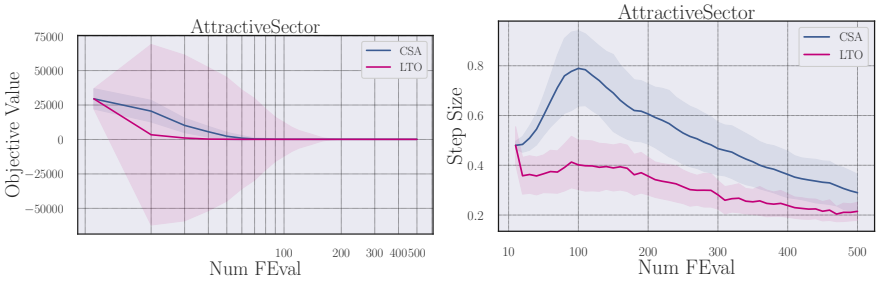


Fig. 3: Step-size adaptation comparison of CSA to that of our learned policies with a sampling rate of 0 and 0.3 on 10 BBOB functions.

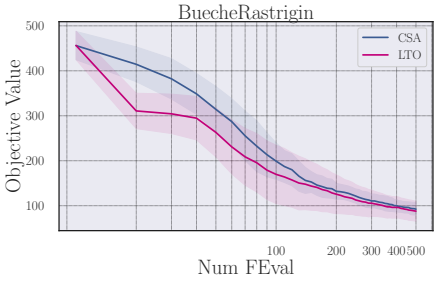
G Transfer to Unseen Test Functions

In this section we provide additional plots for Section 5.4 of the main paper. In the following plots we show the resulting optimization trajectory (left) when using the baseline and learned policy (right) on the corresponding function. We can see that, especially in the beginning, our learned policy learns to use smaller step-size values than CSA.

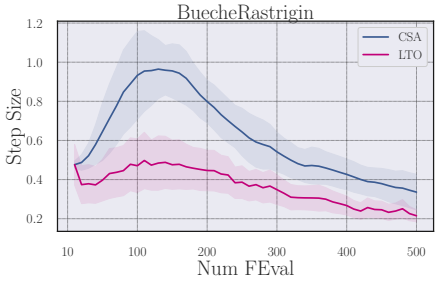


(a) Optimization Trajectory

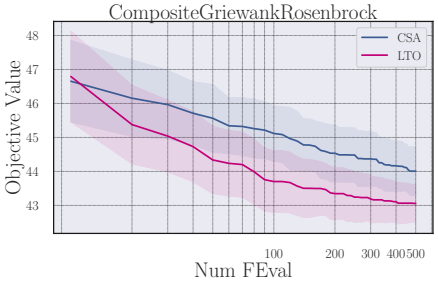
(b) Step-Size Policy



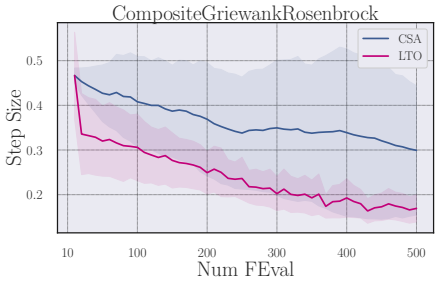
(c) Optimization Trajectory



(d) Step-Size Policy

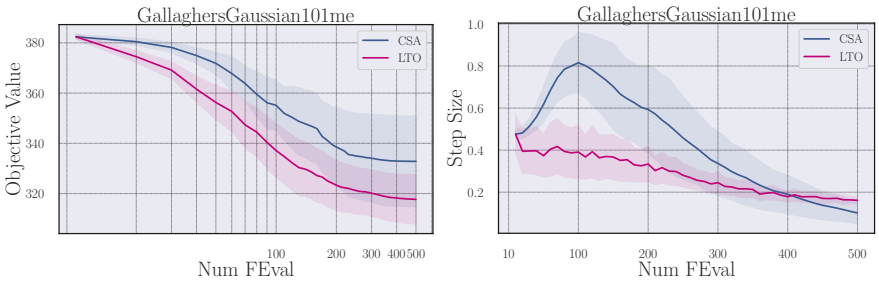


(e) Optimization Trajectory



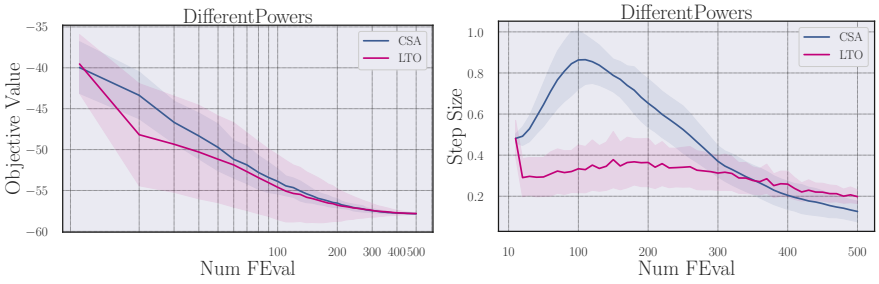
(f) Step-Size Policy

Fig. 4: Optimization trajectories/Step-Size policies of CMA-ES using CSA (blue) and our learned policy (magenta) on 12 unseen test functions.



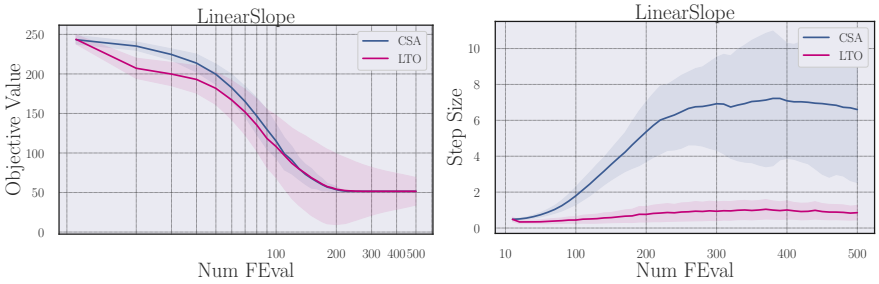
(g) Optimization Trajectory

(h) Step-Size Policy



(i) Optimization Trajectory

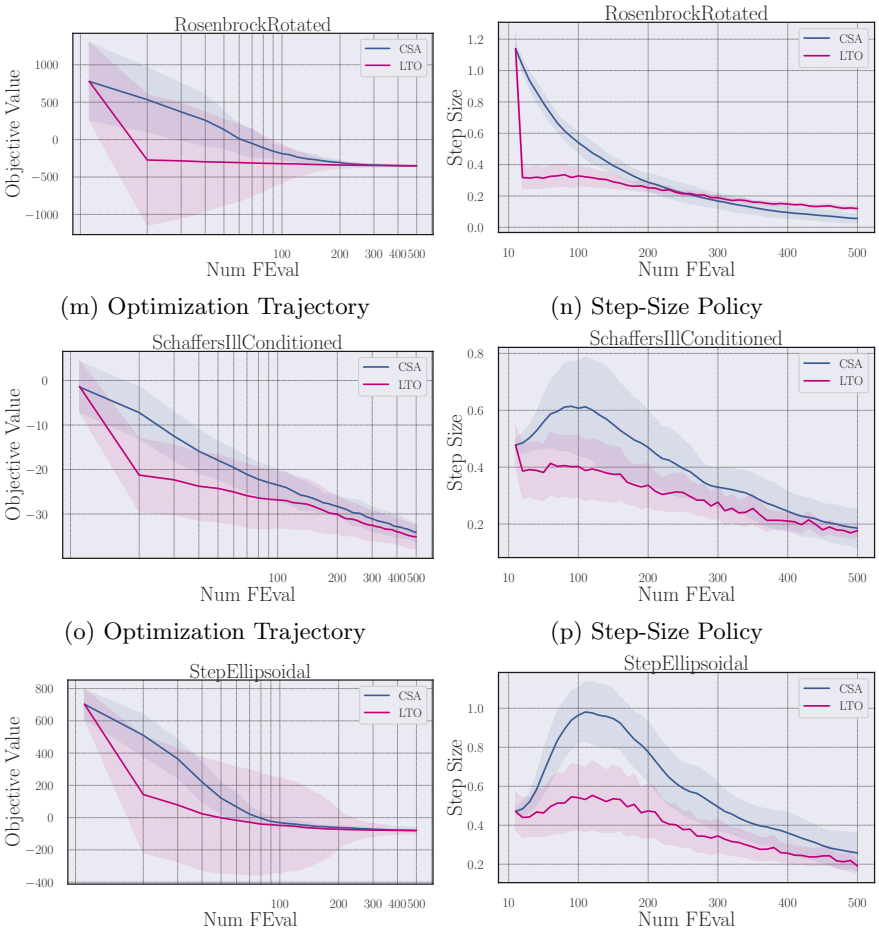
(j) Step-Size Policy



(k) Optimization Trajectory

(l) Step-Size Policy

Fig. 4: Optimization trajectories/Step-Size policies of CMA-ES using CSA (blue) and our learned policy (magenta) on 12 unseen test functions.

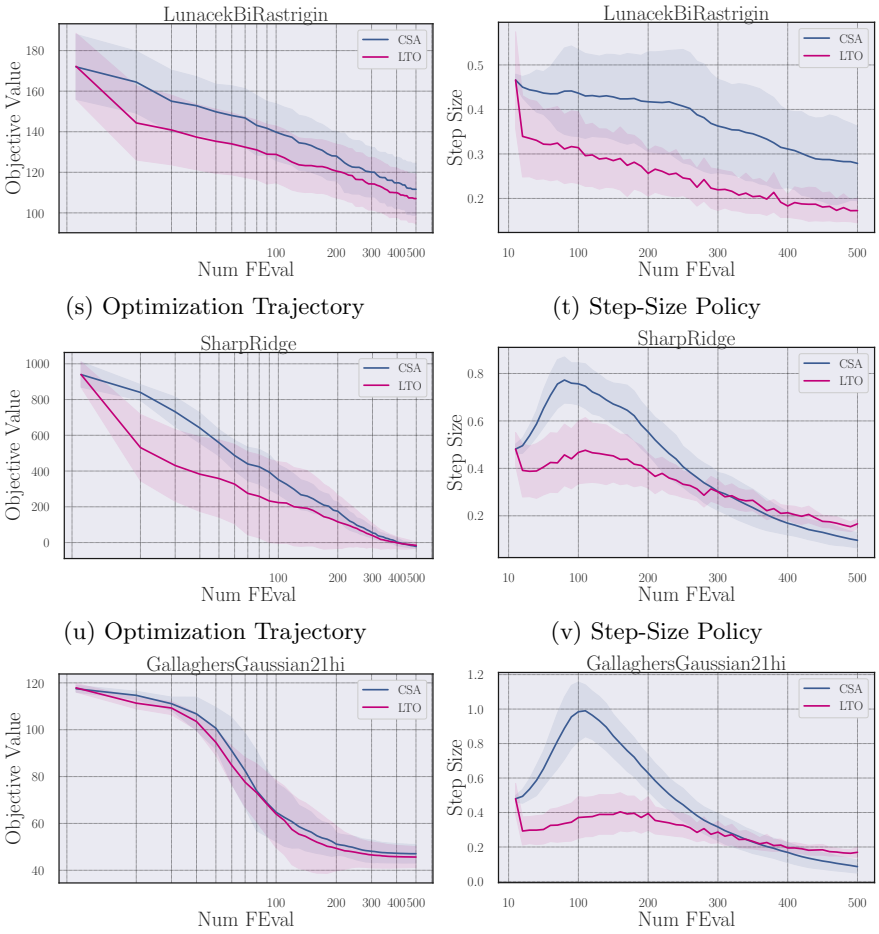


(m) Optimization Trajectory (n) Step-Size Policy

(o) Optimization Trajectory (p) Step-Size Policy

(q) Optimization Trajectory (r) Step-Size Policy

Fig. 4: Optimization trajectories/Step-Size policies of CMA-ES using CSA (blue) and our learned policy (magenta) on 12 unseen test functions.



(s) Optimization Trajectory (t) Step-Size Policy

(u) Optimization Trajectory (v) Step-Size Policy

(w) Optimization Trajectory (x) Step-Size Policy

Fig. 4: Optimization trajectories/Step-Size policies of CMA-ES using CSA (blue) and our learned policy (magenta) on 12 unseen test functions.