Sample Efficient Bayesian Optimization for Policy Search: Case Studies in Robotics and Education

Rika Antonova

CMU-RI-TR-16-40

July 2016

The Robotics Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

Submitted in partial fulfillment of the requirements for the degree of Master of Science in Robotics.

Thesis Committee:
Emma Brunskill (Chair)
Christopher G. Atkeson
Akshara Rai (Student Member)

Keywords: Bayesian optimization, sample efficiency, policy search, bipedal locomotion, intelligent tutoring systems

Abstract

In this work we investigate the problem of learning adaptive strategies, called policies, in domains where evaluating different policies is costly. We formalize the problem as direct policy search: searching the space of policy parameters to identify policies that perform well with respect to a given objective. Bayesian Optimization is one method suitable for such settings, when sample/data efficiency is desired. We use this method as a starting point and present approaches that further improve sample efficiency.

To take advantage of domain knowledge we propose an approach for constructing a domain-specific kernel. This construction utilizes a simulator of the underlying model dynamics, but does not require the simulator to perfectly capture the dynamics. We demonstrate the success of this approach on the case of learning bipedal locomotion policies and outline conditions necessary for a similar approach to be useful in other domains.

In some settings model-based approaches are weakened by the lack of domain knowledge. However, the task structure could be used to improve sample efficiency in a model-free way. We demonstrate how this can be achieved for the case of learning optimal stopping policies. We propose a resampling approach that reuses samples/trajectories for model-free off-policy evaluation. "Off-policy" means we can reuse previously collected samples/trajectories to evaluate new policies. This allows to significantly reduce the number of costly samples collected from the environment during optimization, while providing a way to evaluate a large number of stopping policies.

For our experiments we consider two domains where policy evaluation is costly: bipedal locomotion and intelligent tutoring systems. For locomotion we construct a domain specific kernel and use it when optimizing control parameters for a recently developed neuromuscular model in simulation. We demonstrate that our approach substantially reduces the number of costly trials. To evaluate our model-free resampling approach we turn to the domain of education. Here we consider the problem of inferring how many instructional exercises are enough to achieve a learning objective. We show that resampling offers improvements over standard Bayesian Optimization in simulation, and is also effective on real interactions with students/participants.

Acknowledgements

Thanks to Emma Brunskill for inspiration and guidance for the part of this work on intelligent tutoring systems. This work was supported in part by the grants Emma obtained: Google Research Award and ONR Young Investigator Award. Also thanks to Joe Runde and Dexter Min Hyung Lee for collaboration on adaptive tutor experiments. More generally, thanks to Emma and her lab for creating a welcoming and supportive work environment.

The part of this work on Bipedal Locomotion grew out of a joint project with Akshara Rai. Many thanks to Akshara for developing determinants of gait metric, setting up the simulator, creating an illustration and summary of the neuromuscular model. Also thanks to Chris Atkeson and Hartmut Geyer for giving Akshara their feedback and suggestions for the project.

And of course, many thanks to my supportive friends and family:

globally – my parents; my favorite mathematician and caring partner, Maksim Maydanskiy; friends who kept in touch despite the geographical distance

locally – thanks to Heather Jones, who introduced me to the group of friends in Pittsburgh called "the dinner train" – it was great to have local friends too!

Contents

List of Figures

1 Introduction

In this work we focus on the problem of developing sample-efficient approaches for learning adaptive strategies, called policies. In domains with high cost of evaluating the effectiveness of any given policy, it is beneficial to evaluate the performance of as few alternative policies as possible when searching for the optimum (that is, to take as few "samples" as possible – hence the term "sample-efficient"). We formalize the problem as direct policy search: searching the space of policy parameters to identify policies that perform well with respect to a given objective. Bayesian Optimization is one sample-efficient method that has been found to work well for such settings. However, after conducting several experiments in the challenging domains of robotics and education, we find the need for further improvements. So we develop approaches to ensure the method does not degrade in higher (5+) dimensional spaces and enable finding well-performing policies fast – in under 25-60 trials (policy evaluations).

We propose an approach for constructing an informed kernel by using a simulator that models the dynamics of the underlying domain. This construction can be done with an imperfect simulator and in settings slightly deviating from those encountered during optimization. Bayesian Optimization can take advantage of such a domain specific kernel, which provides an informed similarity between different policies. This helps the optimization to effectively separate good regions of the parameter space from bad regions, making it more sample efficient. We demonstrate the success of this approach on the case of learning bipedal locomotion policies and outline conditions necessary for a similar approach to be useful in other domains.

Model-based approaches, however, can be weakened in settings with lack of domain knowledge. Yet in some cases the task structure could be used to improve sample efficiency in a model-free way. We demonstrate how this can be achieved for the case of learning optimal stopping policies. We develop a resampling approach that reuses samples/trajectories for off-policy evaluation. This approach does not require a simulator or specific domain knowledge. Instead off-policy evaluation is enabled by the structure of the optimal stopping problem. This is possible in domains where a trajectory collected using a particular stopping policy could be used to evaluate other policies that would dictate to stop earlier. Our resampling approach allows to significantly reduce the number of costly samples collected from the environment, while providing a way to evaluate a large number of parameters for stopping policies. We present results for the case of optimizing instructional policies in the domain of education, both on simulated student data and using real-world students/participants recruited via Amazon Mechanical Turk.

1.1 Domain Details and Overview of Experiments

We consider two domains where samples are particularly costly: the domain of robotic locomotion (where evaluating locomotion policy parameters requires either executing the policy on an actual robot in real time, or costly high-fidelity simulations) and the domain of intelligent tutoring systems for education (where evaluating instructional policy for a course or a skill requires time and attention of a student over several hours or even weeks).

To facilitate learning bipedal locomotion policies with fewer costly trials (policy evaluations), we employ Bayesian Optimization to learn optimal reflex parameters for neuromuscular models described in [27] and convert them to robot control parameters. We start with 2D 7-link simulated robot with hip, knee and ankle actuation. We formulate a cost function which incorporates components like distance and time walked, speed and cost of transport (energy used), and use Bayesian Optimization to sequentially select and evaluate various sets of policy parameters to minimize this cost function. Our 16-dimensional policy search space proves to be challenging for the standard

Bayesian Optimization used in prior work, with performance not far from uniform random search for the first 100 trials. So we introduce several novel extensions to improve sample efficiency. First we implement a simple approach of constructing priors using fast coarse simulations, which provides an easy way to get an initial improvement. We then propose a method for constructing an informed kernel. Bayesian Optimization can take advantage of a domain specific kernel, which could provide informed similarity between different policies (this helps identifying promising regions of the search space faster). Our kernel captures similarity between policies using characteristics of gaits induced by these policies. We pre-compute these characteristics for 100K points by running short simulations on flat terrain for a grid of points (a point represents a set of control policy parameters, and we simulate the corresponding policies for 5 seconds each). We demonstrate that the resulting kernel substantially reduces the number of trials (policy evaluations) necessary when optimizing cost functions on other terrains, and when simulating a robot with changes in mass and inertial properties. This signifies that our kernel improves sample efficiency in simulated conditions deviating from the setting in which it was generated. Hence it is a potential candidate for improving optimization done in real time on the hardware. In such cases the real system resembles but does not exactly match the conditions of the simulator, and a sample-efficient online approach is more desirable than the alternative of directly using the policy parameters optimized to convergence only in simulation (since these tend to overfit to the simulator dynamics).

For the domain of intelligent tutoring systems, we consider the problem of learning optimal stopping policies that dictate how many instructional exercises are enough to achieve a learning objective. We start by presenting the benefits of doing direct policy search (over first fitting a model and then constructing a policy given that model, as is particularly common in educational data mining). Then we introduce a resampling method, BO-RESAMPLE, to improve sample-efficiency of Bayesian Optimization for the case of learning optimal stopping policies. Our method allows off-policy evaluation of alternate policies (policies that would have stopped earlier than those evaluated so far). This allows constructing more informed posteriors with fewer costly evaluations. We compare the performance of BO-RESAMPLE with standard Bayesian Optimization and show that our approach is able to find effective policies with substantially fewer costly evaluations. Furthermore, we discuss a comparison with a recently developed Model-based Bayesian Optimization Algorithm (MBOA) [31], which facilitates learning with fever policy evaluations when information about the model of the student is available. BO-RESAMPLE learns slower than MBOA when model assumptions of MBOA are correct, but outperforms it if the student behavior does not match the model assumptions. Because of the limitations of the current best student modeling approaches and the limited amount of data on student learning and behavior, it is frequently not possible to get a good fit of the model for a given intelligent tutoring task. Hence we conclude that in such cases of model mismatch and/or insufficient data, our direct policy search approach with resampling provides a more robust and sample-efficient alternative.

2 Background and Motivation

2.1 Policy Search and Parameter Optimization

One of the standard and concrete ways to formulate a policy search problem is to consider a class of policies generally known to be effective for a given domain, then search the space of policy parameters for the set(s) of parameters that minimize a certain criterion (also called cost or objective function). Then, formally the goal is to find:

$$\boldsymbol{x}^* = \arg\min_{\boldsymbol{x}\in\mathbb{R}^d} f(\boldsymbol{x})$$

with $\boldsymbol{x} \in \mathbb{R}^d$ representing a vector of policy parameters, function f being the cost or objective function incorporating various domain-dependent performance criteria, and $\boldsymbol{x}^*$ representing a vector of policy parameters that achieve the minimum cost.

During policy search each trial (cost function evaluation) entails executing a policy initialized with a parameter vector $\boldsymbol{x}$ to obtain cost $f(\boldsymbol{x})$ (and in stochastic domains we only obtain a noisy estimate of $f(x)$ from a single trial, not the perfect knowledge of f at $\boldsymbol{x}$). Along with the cost, we also obtain a trajectory which corresponds to the behavior of the system when policy with parameters $\boldsymbol{x}$ is used. For various tasks and domains these trajectories take different forms. For example, when learning control policies for locomotion, a trajectory would correspond to the walking (or falling) behavior of the robot (or the simulated model). In the domain of education the trajectory would correspond to observations of student behavior (e.g. correct/wrong answers to practice and assessment problems). Most real-world domains are stochastic, so executing the same policy with the same parameters would likely result in different trajectories. In the domain of locomotion this would correspond, for example, to unforeseen/unmodelled changes in the surrounding environment or hardware. In education this would correspond to interacting with a different student. Frequently the cost $f(\boldsymbol{x})$ can be computed directly from the corresponding sampled trajectory. For locomotion, the relevant information includes time and distance walked, center of mass locations (e.g. to detect falls) – from these we could compute a cost that indicates whether the trajectory corresponded to an effective walking attempt. In education, the cost could be computed, for instance, by taking into account the number of problems given during instruction and performance on assessment problems.

Optimizing f analytically is usually infeasible, since the precise dependence of the function on policy parameters is not known. However, a variety of algorithms for optimization of an unknown function can be applied to this problem. These algorithms vary in assumptions they make about the objective function. The least restrictive, but also least efficient algorithms include grid search, pure random search, and various evolutionary algorithms. Gradient-based algorithms can be used for this problem if numerical gradients can be computed or approximated effectively. However, standard gradient-based algorithms (such as stochastic gradient descent) find only a local optimum, hence several restarts would be needed for the search to be global. Second-order methods could be used to speed up the search, but these methods might require additional evaluations of f to compute numerical gradients.

Bayesian Optimization does not require require additional samples to reason about the next step, and efficiently models the cost function landscape globally. Moreover, the sequential nature of Bayesian Optimization allows this algorithm to use maximum information about all the policy evaluations done so far, while non-sequential methods (e.g. evolutionary strategies) evaluate several policies before adjusting decisions on which set of policies to try next. Hence Bayesian Optimization is a good candidate for sample-efficient global policy search. This conclusion has also been reached by Calandra *et al.* in [6], which presented more extensive arguments and comparisons of various optimization methods for policy search for locomotion.

2.2 Overview of Bayesian Optimization

Bayesian Optimization provides a framework for sequential global search, well suited for sample efficient derivative-free function optimization (introduced by [15] and originally referred to as Efficient Global Optimization; see [26], [4] for recent overviews). The goal is to find a vector $\boldsymbol{x}$ that optimizes a given objective function $f(\boldsymbol{x})$, while executing as few costly evaluations of f as possible. The optimization starts with initializing the prior (which could be uninformed) capturing the prior uncertainty over the value of $f(\boldsymbol{x})$ for each vector $\boldsymbol{x}$ in the domain. Then an auxiliary optimization function (called acquisition function) is used to sequentially select each next $\boldsymbol{x}$, and $f(\boldsymbol{x})$ is evaluated (the corresponding experiment or task is performed in the underlying environment). The aim of the acquisition function is to achieve effective tradeoff between exploration and exploitation: select enough points for which the posterior estimate of the objective f is promising, while also selecting some points with the aim to maximally decrease the uncertainty about f in the posterior. One of the most widely used acquisition functions is Expected Improvement (EI) [22]. EI selects $\boldsymbol{x}$ to maximize expected improvement over the value of the best result obtained so far. For minimization problems:

$$\boldsymbol{x}^- = \underset{x_i \in x_{1:t}}{\arg\min}\, f(\boldsymbol{x}_i), \text{where } \boldsymbol{x}_i \text{ are points for which objective } f \text{ has been evaluated so far}$$

$$EI(\boldsymbol{x}) = \begin{cases} [f(\boldsymbol{x}^-) - \mu(\boldsymbol{x})]\Phi(Z) + \sigma(\boldsymbol{x})\phi(Z) & \text{if } \sigma(\boldsymbol{x}) > 0 \\ 0 & \text{otherwise,} \end{cases}$$

μ, σ are posterior mean, variance; $Z = (f(\boldsymbol{x}^-) - \mu(\boldsymbol{x}))/\sigma(\boldsymbol{x})$; ϕ, Φ are PDF and CDF of standard normal distribution. Another popular alternative is Upper Confidence Bound (UCB) or, for minimization problems, Lower Confidence Bound (LCB) acquisition function [28]. LCB balances the mean μ and variance σ of the posterior by selecting points which minimize $\mu(\boldsymbol{x}) - \alpha\sigma(\boldsymbol{x})$.

A common way to model the prior and posterior for f is by using a Gaussian Process:

$$f(\boldsymbol{x}) \sim \mathcal{GP}(\mu(\boldsymbol{x}), k(\boldsymbol{x}_i, \boldsymbol{x}_j)),$$

with mean function μ and kernel k. The prior mean function $\mu(\boldsymbol{x})$ can be set to 0 if no relevant domain-specific information is available. The kernel $k(\boldsymbol{x}_i, \boldsymbol{x}_j)$ encodes how similar f is expected to be for two inputs $\boldsymbol{x}_i, \boldsymbol{x}_j$: points close together are expected to influence each other strongly (in the sense of evidence about $f(\boldsymbol{x}_i)$ influencing posterior for $f(\boldsymbol{x}_j)$ and vice versa), while points far apart would have almost no influence. The most widely used kernel is the Squared Exponential kernel:

$$k_{SE}(\boldsymbol{x}_i, \boldsymbol{x}_j) = \sigma_k^2 \exp\Big(-\frac{1}{2\boldsymbol{l}^2}\|\boldsymbol{x}_i - \boldsymbol{x}_j\|^2\Big),$$

with hyperparameters: σ_k^2 – signal variance, $\boldsymbol{l}^2$ – vector of length scales.

A Gaussian Process conditioned on evidence represents a posterior distribution for f. In many settings only noisy estimates can be obtained when evaluating $f(\boldsymbol{x})$. Hence, at each next step t+1, the evidence $\boldsymbol{y} \in \mathbb{R}^t$ is assumed to consist of the noisy estimates $\boldsymbol{y}_i = f(\boldsymbol{x}_i) + \epsilon_{\mathcal{N}}$ (i=1...t), with $\epsilon_{\mathcal{N}} \sim \mathcal{N}(0, \sigma_{noise}^2)$. When deciding what point $\boldsymbol{x}_{t+1}$ to consider next, we can use the fact that joint distribution of observations so far and posterior value f_{t+1} of a new point $\boldsymbol{x}_{t+1}$ is a multivariate Gaussian. This predictive posterior $P(f_{t+1}|\boldsymbol{x}_{1:t}, \boldsymbol{y}, \boldsymbol{x}_{t+1}) \sim \mathcal{N}\big(\mu_t(\boldsymbol{x}_{t+1}), cov_t(\boldsymbol{x}_{t+1})\big)$, with mean and covariance (from [23]):

$$\mu_t(\boldsymbol{x}_{t+1}) = \boldsymbol{k}^T[\boldsymbol{K} + \sigma_{noise}^2\boldsymbol{I}]^{-1}\boldsymbol{y}$$

$$cov_t(\boldsymbol{x}_{t+1}) = k(\boldsymbol{x}_{t+1}, \boldsymbol{x}_{t+1}) - \boldsymbol{k}^T[\boldsymbol{K} + \sigma_{noise}^2\boldsymbol{I}]^{-1}\boldsymbol{k},$$

where $\boldsymbol{k} \in \mathbb{R}^t$, with $\boldsymbol{k}_i = k(\boldsymbol{x}_{t+1}, \boldsymbol{x}_i)$; $\boldsymbol{K} \in \mathbb{R}^{t\times t}$ with $\boldsymbol{K}_{ij} = k(\boldsymbol{x}_i, \boldsymbol{x}_j)$; $\boldsymbol{I}$ is an identity $\in \mathbb{R}^{t\times t}$.

2.3 Bayesian Optimization in Robotics

Bayesian Optimization (BO) with Gaussian Process likelihood and closely related methods have been recently applied to several Robotics domains. Krause *et al.* [17] developed an approach utilizing Gaussian Processes and the principle of optimizing mutual information for solving sensor placement problem. Martinez-Cantin *et al.* [21] used BO for online path planning for optimal sensing with a mobile robot. Lizotte *et al.* [19] used a closely related approach of Bayesian Gaussian Process Regression to optimize the gait on a quadruped robot and showed that this approach required substantially fewer gait evaluations than state-of-the-art local gradient approaches.

More specific to the domain of bipedal locomotion, Calandra *et al.* very recently used BO to efficiently find gait parameters that optimize a desired performance metric [5]. In a set of experiments with a small bipedal robot, they demonstrated that good walking gaits could be efficiently found by BO. The robot used in their experiments was a small bipedal walker. Eight parameters of the controller were identified as important for the resulting gait, yielding an 8-dimensional optimization problem. These gait parameters consisted of four threshold values of the finite state machine of the controller (two for each leg), and four other control signals were applied during extension and flexion of knees and hips. The authors noted the need to develop a sample-efficient algorithm that could optimize control parameters on the current hardware directly. This is desirable, since a set of parameters performing well with one set of motors could be ineffective with a different set of motors after maintenance (because of, for example, slight differences in the mechanical properties of the new motors). In their experiments, BO with Gaussian Process likelihood and the squared exponential kernel was used to optimize a simple objective function of maximizing the speed of the robot. Several acquisition functions were evaluated (Probability of Improvement, Expected Improvement, UCB/LCB) as well as automatic vs. manual optimization of BO hyperparameters. For their 8-dimensional problem, the authors reported achieving stable walking in less than 30 trials (policy evaluations).

While these previous results using Bayesian Optimization for optimizing locomotion controllers are encouraging, it is not immediately clear whether BO would be as successful in finding good policies for higher-dimensional problems. Calandra *et al.* mentioned that only around 1% of the parameter space they considered led to walking gaits, and we have observed similar difficulties in finding promising regions of the policy parameter space in our experiments with higher-dimensional policy search. Hence one of the questions that needs to be addressed: would BO be effective when learning control policies for a full-scale humanoid robot?

Another important question is: how does BO compare to approaches like Covariance Metrics Adaptation Evolution Strategy (CMA-ES) [27], which also has been previously used for gait optimization (for example in [13])? While Calandra *et al.* mentioned a number of state-of-the-art algorithms used for optimizing control policies in robotics, they only presented comparison of their BO experiments with random and grid search. So in our experiments we do address the question of how BO compares with CMA-ES, and evaluate simple modifications of CMA-ES to make it a competitive alternative and hence a robust baseline for evaluating sample-efficiency of BO.

2.4 Kernels for Sequential Decision Making

As described in Section 2.2, the kernel (or the covariance function) $k(\boldsymbol{x}_i, \boldsymbol{x}_j)$ captures how similar the objective function f is expected to be for inputs $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$. In the frequent case of using the squared exponential kernel (also defined in Section 2.2), the similarity is a function of the Euclidean distance between $\boldsymbol{x}_i, \boldsymbol{x}_j \in \mathbb{R}^d$. When BO is used for policy search, points $\boldsymbol{x}_i, \boldsymbol{x}_j$ represent policy parameters. First we note that it is a-priori unknown how sensitive the cost function f is to changes in each of the policy parameters. It might turn out that the cost function is sensitive to changes in

some parameters, but not others. Fortunately, BO allows us to learn the average responsiveness of f to changes in each input dimension separately. For the squared exponential kernel this is captured by learning length scale hyperparameters automatically with hyperparameter optimization done after each (or after a batch) of policy evaluations. Such covariance functions with learnable length scale parameters are called kernels with automatic relevance determination (ARD). However, this is not sufficient in the cases where particular policy parameterization results in creating small regions of the space with well-performing policies and large regions of the space with ineffective policies. In such cases a single length scale parameter is not sufficient to describe the response of the objective function across the whole (even single) dimension. Re-parameterizing (altering the ways policies are encoded) could help, however it is not trivial to design good parameterizations without further knowledge, analysis and/or data.

One alternative is to use a kernel that specifically leverages the structure of the output generated by executing policies, namely the resulting trajectories or behavior. Intuitively, a kernel that can better encode similarity among policies will be more sample-efficient, since it will be better able to generalize across policies with similar performance. The Behavior-Based Kernel (BBK) of [31] is one such kernel approach that leverages structure in the trajectories generated by the evaluated policies. BBK was developed for stochastic policies and tested on several simplified benchmark Reinforcement Learning domains (Mountain Car task, Cart-pole task, 3-link planar arm domain, bicycle balancing). BBK offered a way to define kernel influences by looking at policy-induced behavior instead of relying on Euclidean distanced between points representing policy parameters. While BBK could help in settings where computing a trajectory for a given set of policy parameters is inexpensive, in the setting of robotic locomotion computing a full trajectory requires running a costly simulation or executing the policy on the real hardware. Hence it is not feasible to use this approach in the kernel. Nonetheless, the idea of using auxiliary information in the kernel is promising, if this information can be pre-computed for a large portion of the policy space and made available during online optimization. We describe our approach to constructing such informed kernel in Section 3.4. Our kernel effectively incorporates domain knowledge available in bipedal locomotion without the need for computing full trajectories, but instead using behavior information effectively from only a short part of the trajectories.

In contrast, for the problem we consider in the domain of stopping policies for tutoring systems, the trajectories can be easily computed for each set of policy parameters. However, the class of policies we consider are deterministic and hence BBK is not directly applicable to this setting. Nonetheless, an alternative closely related approach could be developed – we summarize our initial explorations with this in Future Work (Section 5.2).

3 Optimization for Bipedal Locomotion

Inspired by the recent work on applying Bayesian Optimization to bipedal locomotion discussed in Section 2.3, we aim to optimize parameters for more complex policies, which can lead to more stable and human-like walking patters. However, as we demonstrate in our experiments discussed below, the higher dimensional search space is challenging for standard Bayesian Optimization (with performance not much better than uniform random search).

Our first improvement is to create a fast prior for Bayesian Optimization. We collect this prior using 100 short (5 second) simulations, then use it when running optimization on further 100 trials, which simulate full-length (100 second) trajectories. Section 3.3 describes the details of this approach.

Section 3.4 describes a more powerful approach we developed to allow Bayesian Optimization to take advantage of a domain specific kernel, where the kernel is constructed in a way that supports transfer learning across variations of the same domain. We present Determinants of Gait (DoG) kernel, which uses gait characteristics to create an appropriate similarity metric. Under this metric, policies that generate walking gaits are closer together and further away from policies that result in a fall. This helps the optimization to effectively separate good regions of the parameter space from bad regions, making it more sample efficient. We pre-compute this kernel on a grid of 100K points by running short simulations on flat terrain. We demonstrate that this can substantially reduce the number of evaluations needed to optimize cost functions on different terrains, with modeling disturbances. This signifies that our kernel helps improve sample efficiency in conditions different from the setting in which it was generated. Potentially, we can generate this kernel in a simulation and use it to do optimization for actual robots.

In the following sections we present the details of the neuromuscular model we use for modeling bipedal locomotion and discuss the parameters of the model we chose to optimize. Then we introduce our experimental setup and argue how it can help anticipating discrepancies between the simulator and the real-world. We then present a more detailed description of fast prior approach and DoG kernel construction, and discuss experimental results we obtained in simulation.

3.1 Neuromuscular Models

We use neuromuscular model policies introduced by Geyer *et al.* from [12] as the controller for a 7-link planar model. This model approximates muscle dynamics and human-inspired reflex pathways to generate joint torques, with the aim to produce gaits similar to human walking. More details about the neuromuscular model are available in a recent overview in [27]. Though originally developed for explaining human neural control pathways, these controllers have recently been applied to robots and prosthetics, for example in [29] and [30].

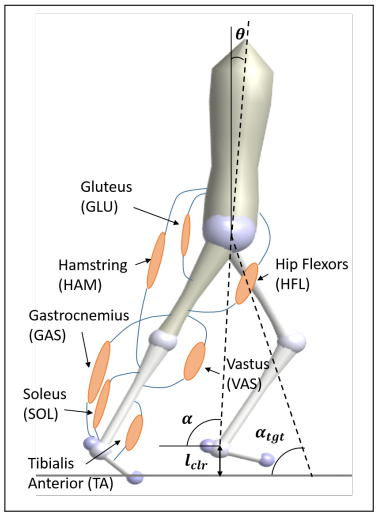
3.1.1 Stance and Swing Control

Each leg is modeled to be actuated by 7 muscles: soleus (SOL), gastrocnemius (GAS), vastus (VAS), hamstring (HAM), tibialis anterior (TA), hip flexors (HFL) and gluteus (GLU) illustrated in Figure 1. These muscles produce torques about the hip, knee and ankle via local feedback laws. The resulting torques are functions of joint angle and force exerted, and the force is a function of the state of a muscle m and muscle stimulus S_m, which in general has the form: $S_m(t) = S_{0,m} + G_m \cdot F_m(t - \Delta t_m)$, where $S_{0,m}$ is the pre-stimulus, G_m is the feedback gain and $F_m(t - \Delta t_m)$ is the time-delayed force. We aim to optimize a subset of gains G_m. For stance this includes: $G_{GAS}, G_{GLU}, G_{HAM}, G_{SOL}$ (positive force feedback gain on GAS, GLU, HAM, SOL), G^{TA}_{SOL}(negative force feedback from SOL on TA), G_{TA}, G_{VAS}, G_{GLU} (positive length feedback on TA, VAS, GLU) – this feedback generates compliant leg behavior and prevents the knee from overextending in stance. Detailed description of

these can be found in Section II.A of [12]. To balance the trunk, feedback of the torso angle is added to the GLU using relationship $S_{GLU}(t) = k_p^{torso}(\theta_{desired} - \theta) - k_d^{torso}\dot{\theta}$, which involves two gains we optimize: k_p^{torso} and k_d^{torso}. Description of this part of the model can be found in Section II.B of [12].

The swing is controlled by three main components – target leg angle, leg clearance and hip control. Target leg angle is the direct result of the foot placement strategy, which is presented in [33]. Section 3.3 of [33] describes the feedback law for the desired leg angle $\alpha_{tgt} = \alpha_0 + c_d d + c_v v$, where α_0 is a default leg angle, d is the distance between the stance leg and the center of mass (CoM) and v is the velocity of the center of mass. In our optimization we include α_0, gains c_d (gain on the distance between stance leg and CoM) and c_v (gain on the velocity of the CoM). Leg clearance is a function of the desired leg retraction during swing, which has been shown to be crucial for stable walking and running in [25]. The knee is actively flexed until the leg reaches the desired leg clearance height and then its held at this height using a damping torque, until the leg reaches a threshold leg angle. At this point, the knee is extended and allowed to reach the target leg angle, without over-extending. Details of this control can be found in [8]. In our experiments, we found that the control is relatively insensitive to the individual gains of this set-up, but the target leg clearance l_{clr} affects the stability and metabolic cost, so we include it in our optimization. A third part of the control involves maintaining a desired leg angle by applying a hip torque. This dynamics is described in Section III.A of [8]. We focus on optimizing k_p^{swing} and k_d^{swing} for hip control $\tau_{hip}^{\alpha} = k_p^{swing}(\alpha_{tgt} - \alpha) - k_d^{swing}(\dot{\alpha})$.

Figure 1. Neuromuscular model.



Altogether, the above yields a 16-dimensional optimization problem to find optimal values for $G_{GAS}, G_{GLU}, G_{HAM}, G_{SOL}, G_{SOL}^{TA}, G_{TA}, G_{VAS}, G_{GLU}, k_p^{torso}, k_d^{torso}, \alpha_0, c_d, c_v, l_{clr}, k_p^{swing}, k_d^{swing}$.

3.2 Experimental Setup: Optimization with Simulated Model Disturbances

3.2.1 Generating Simulated Model Disturbances

While in theory, and in simulators, robotic models have precise specifications and well-understood dynamics, most real robots have poor dynamic models, as well as unmodelled disturbances, like friction, non-rigid dynamics, etc, which frequently make simulations a poor representation of the robot's real state. Parameters like the mass and inertia of the rigid body dynamics equations for the real robots are usually not known precisely, since a real robot is not just a combination of the links, but also includes actuators, screws, wirings and other components. Even if the mass of each of these components could be measured accurately, their relative placement could still affect the mass distribution on the robot. There has been a lot of work in identifying dynamic models of robots reliably, for example in [1]. Such methods can help bring simulators closer to modeling the real robot, though there are other discrepancies like non-rigid dynamics, friction and actuator dynamics, which are still hard to model. As a result, often controllers that work well in simulation lead to poor performance on real robots. To accommodate such cases, we would like to develop optimization techniques that quickly adapt to closely related but different settings of the real hardware – algorithms that can search for an effective set of control parameters for a simulated environment, and then can find new optimal solutions with few costly trials involving the real hardware.

The first step in this direction is to develop an experimental setup where we could test whether prior optimization done in one setting could be a useful start for sample-efficient optimization in a related but perturbed setting. Before such tests could be done on real hardware, for our experiments we establish the setup which uses only simulated environments, with related but perturbed dynamics. An initial stage of optimization or any necessary pre-computation is done in a simpler unperturbed simulated setting on a flat ground profile. Then to test if the algorithms are capable of generalizing to unforeseen disturbances, modeling and environmental perturbations, we simulate robot models with mass and inertial disturbances on rough ground profiles. We perturb the mass of each link, inertia and center of mass location randomly by up to 15% of the original value. For mass/inertia we randomly pick a variable from a uniform distribution between $[-0.15,\ 0.15] \cdot M$, where M is the original mass/inertia of the segment. Similarly, we change the location of the center of mass by $[-0.15,\ 0.15] \cdot L/2$, where L is the length of the link. These perturbations are similar to the kind of modeling disturbances seen on a real robot, where the exact values of mass, inertia and center of mass locations are unknown. To generate rough ground we add ground height changes of up to $\pm 6cm$. In our experiments we generate different disturbances and ground profiles for each optimization run. Hence we test a wide range of model and ground disturbances.

3.2.2 Details of Experimental Setup: Cost Function and Algorithms Compared

Since we formulate our optimization problem as searching for an optimal set of control parameters that optimize some cost function, we need to construct a cost function. We design a cost function that varies smoothly over the parameter space, such that parameter sets achieving low cost also achieve stable walking gaits:

$$cost = \frac{0.5}{1+t} + \frac{0.3}{1+d} + 0.1(s - s_{tgt}) + 0.01 c_{tr},$$

where t is seconds walked, d is the final hip position, s is mean speed, s_{tgt} is the desired walking speed (from human data), and c_{tr} is cost of transport computed from an estimate of the metabolic energy required.

In the following sections we compare the performance of several baseline and state-of-the-art optimization algorithms in simulations. Motivated by the discussion in [6], we include the baseline of uniform random search. While this search is uninformed and not sample-efficient, it could (perhaps surprisingly) serve as a competitive baseline in non-convex high-dimensional optimization problems. Theoretically, it provides statistical guarantees of convergence, and practically it can outperform informed algorithms as well as grid search on high-dimensional problems (see Section 2 in [6] for further discussion).

We provide comparisons with the Covariance Matrix Adaptation Evolutionary Strategy (CMA-ES) – an evolutionary algorithm widely used for non-linear non-convex black-box optimization in continuous domains. On convex problems it converges to the global optimum. On more general problems it can get stuck in local minima, however empirically it has been shown to work well for a wide range of problems. Details of this algorithm can be found in [13]. Even though CMA-ES has been used to optimize non-convex problems in high dimensions, it can be slow and not sample efficient and its performance depends substantially on the initial starting point. For example, when used to optimize parameters of a neuromuscular model in [27], it takes 400 generations and $\approx$5,000 function evaluations (reported by the authors to take about a day on a standard desktop processor). Depending on the initial starting point, the performance can be quite different in terms of time taken to converge, as well as the quality of the local optimum found.

For experiments with Bayesian Optimization we explored using two libraries: MOE developed by Yelp [14] and Matlab implementation from [10]. MOE provides a mature stand-alone BO server,

implemented in python and C++, that has been used both in industry and academia. The Matlab implementation from [10] builds on a Gaussian Process library implementation from [23], and was useful to us to avoid cross-language overhead when using Matlab Simulink for running simulator of the robot models.

In our experiments we compute the cost for a set of policy parameters by running Matlab Simulink simulator for the neuromuscular model from [29] (a recent re-implementation of [11]) for 100 seconds, setting initial mass and inertial properties to match the average human parameters described in [32]. All our results are evaluated on rough ground profiles with inertial and mass model disturbances described in Section 3.2.1, and to interpret optimization progress we usually plot the cost of the best policy found so far versus the number of 100-second trials (policy evaluations) performed. We also visualize the walking behavior of several best-performing policies obtained to ensure the low costs correctly correspond to stable walking behavior. Since we are dealing with a highly non-convex function in a 16-dimensional space, it is not practical to obtain the exact global minimum to evaluate how close our results come to finding the best policy. However, we can obtain an estimate for the global minimum for the cost used in our experiments. To get an estimate we ran the best-performing versions of CMA-ES from various starting points until convergence for 50 runs and found that the best policies had a cost of $\approx$0.122. Since the best cost that could be achieved on rough ground with model disturbances is likely to be higher, this gave us and estimate for a lower bound for the optimum (since it was obtained in the less challenging setting).

3.3 Improving Sample Efficiency with Fast Prior

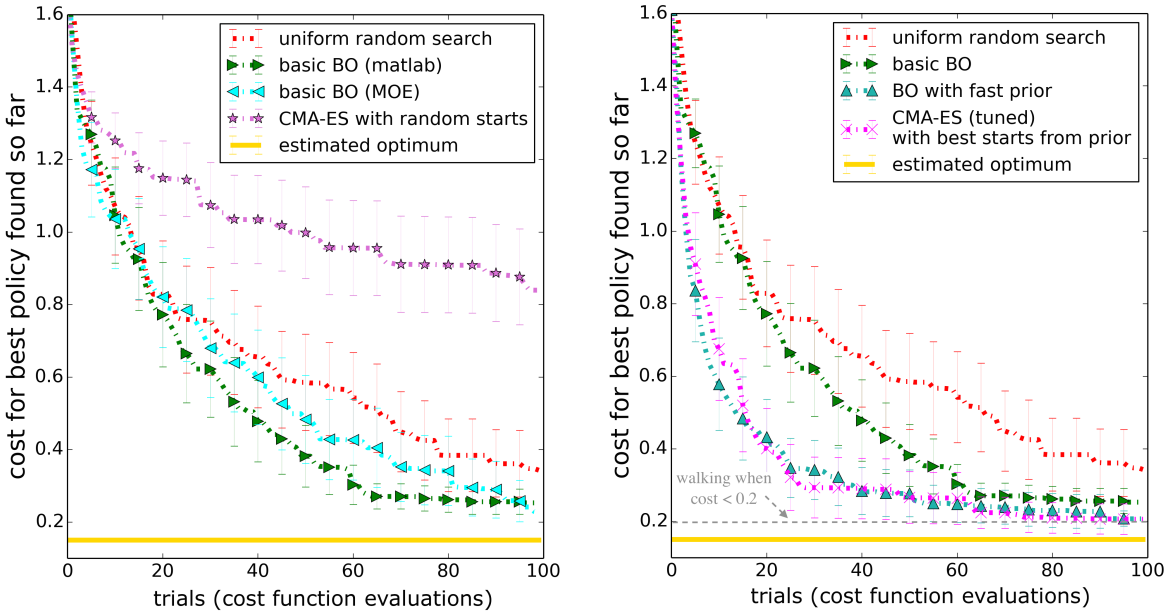
Our initial experiments[1] with basic Bayesian Optimization developed in prior work showed that basic BO was not able to substantially outperform uniform random search in our 16-dimensional policy search space. Left side of Figure 2 illustrates experiments with two Bayesian Optimization libraries mentioned in Section 3.2.2. In both cases we used uninformed prior, the squared exponential kernel and the built-in automatic hyperameter optimization when running BO experiments. We did 50 separate optimization runs (each lasting up to 100 trials) and plotted the mean (and 95% confidence interval) of the cost of the best policy found so far. We observed that BO algorithms offered some improvement over uniform random search, but were not able to reliably find policies that would ensure walking on rough ground profiles in less than 100 trials. We also experimented with CMA-ES, but when ran from randomly selected initial starting points, CMA-ES performed significantly worse than uniform random search in terms of best policy found so far versus number of trials performed.

Since baseline state-of-the-art algorithms like CMA-ES and Bayesian Optimization were not able find effective policies in a sample-efficient way, we worked on developing an improved approach. Bayesian Optimization is well-suited to incorporating noisy prior observations. So we collected a prior of 100 policies evaluated during short 5-second simulations on flat ground using unperturbed model of the robot. We then initialized BO with these points when performing optimization on rough ground profiles with perturbed models. We set higher noise variance level for points collected for the prior, indicating that while the values obtained during short simulations have some useful indication of the cost, the actual costs for the corresponding policies on rough ground with perturbed models simulated for a longer period of time (100 seconds) are expected to differ.

The right side of Figure 2 shows the results for BO with fast prior. We can see that it improves over the performance of basic BO, and is able to find policies that walk on rough terrain within $\approx$95 trials. To see whether fast prior idea could benefit other algorithms, we also ran a tuned version of CMA-ES, for which we first ran 100 5-second simulations, then started CMA-ES from the point that

[1] For experiments in this section we generated more challenging ground profiles with height changes of up to $\pm 4.5cm \cdot x$, where x was the distance walked so far.

Figure 2. Left: Experiments comparing baseline BO and CMA-ES with uniform random search. Right: BO with fast prior versus basic BO and CMA-ES started with best points from fast prior.



obtained the best (estimate of) the cost during this short stage. As shown in Figure 2, CMA-ES is also able to make effective use of the preliminary short simulations.

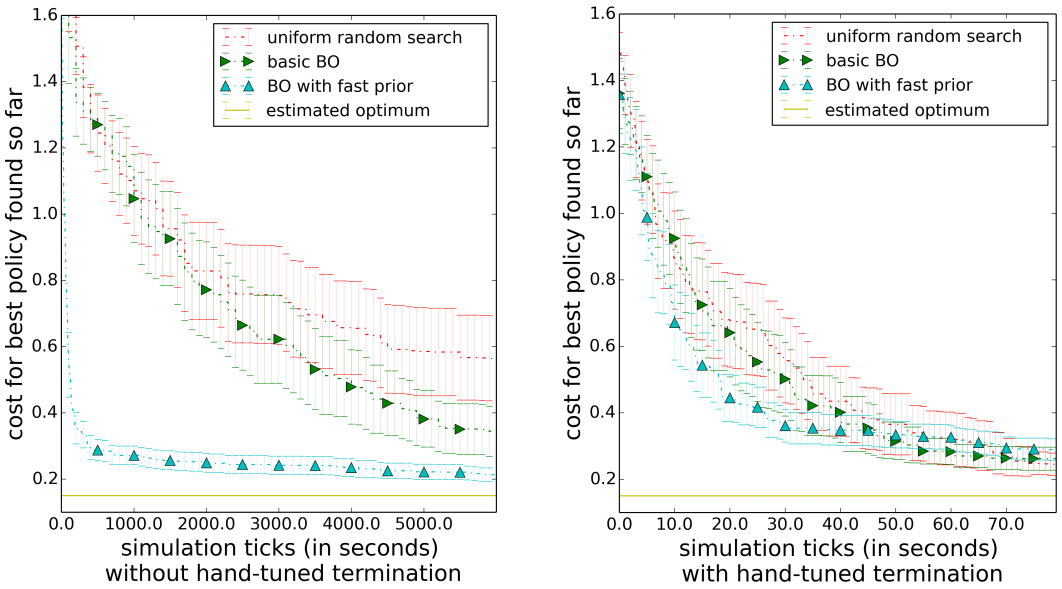
In principle, fast prior approach can be applied to a wide range of domains – those where a short simulation/trial can provide a coarse estimate of how well the policy would perform during longer simulations/trials. In practice – an interesting direction for future work is to apply this method to various domains and see whether it would provide similar improvements as what we observed in our experiments with learning bipedal locomotion policies.

To investigate the advantage of using fast prior in terms of time spent we also plot the costs for best policy found so far versus simulation ticks in Figure 3. The plot on the left shows what would happen in the case where the simulations run for the planned time. For uniform random search and basic BO this means all the costs need to be obtained by evaluating policies for 100s in the simulator, while for BO with fast prior, the points for the prior can be obtained by running the first 100 simulations for only 5 seconds each on flat ground, then using the prior for optimization on rough ground with model disturbances and evaluating each policy for 100 seconds. The end of the x axis on the plot is at 60K seconds, which corresponds to 60 trials (policy evaluations) for uniform random search and baseline BO, and to 100 5s policy evaluations plus 55 full-length 100-second trials for BO with fast prior.

While BO with fast prior would initially seem to be able to offer much improved efficiency in terms of simulation time, we note that in our particular domain various heuristics can be developed to perform early stopping when a policy clearly does not seem promising to yield a stable walk. We implemented such checks, including stopping the simulation when: the center of mass (CoM) is detected to be on the ground (the model falls), joints go beyond joint limits, CoM is detected to be more than $5m$ above the ground (the model attempts to jump, which for this model invariably results in a crash). The plot on the right side of Figure 3 reveals that most of the 100-second simulations could be stopped early, and after 80 seconds (the end of the x axis on the plot) more than 80 policies can be evaluated, with most policies obtaining a very high cost and terminated early.

The conclusion we draw is that for domains without clear potential for early stopping, approaches

Figure 3. Performance comparisons with respect to the number of planned simulation ticks (left) and actual simulation ticks executed after implementing early stopping heuristics (right).



like BO with fast prior could be effective in improving sample efficiency as well as decreasing the amount of time needed for simulation (or operation with real hardware). However, if an effective set of rules for early stopping is available, then the benefits of this approach would not be as profound, because it would not take long to rule out a large number of policies as non-promising fast. This observation inspired us to develop an alternative approach that we describe in the following sections.

3.4 Determinants of Gait (DoG) Kernel for Bipedal Locomotion

In Section 2.4 we motivated the use of kernels that encode the domain knowledge about the behavior induced by policy parameters. We described why in our setting it would be particularly helpful to augment or replace uninformed Euclidean distances in the space of control policy parameters with more informed domain-specific distance metrics. Even though in our domain the full trajectory of the robot is not available without a costly evaluation, we can efficiently simulate the behavior for 3-5 seconds for a large number of different sets of parameters. This capability allows us to develop a domain-specific distance metric that captures some basic bipedal gait measures and can be used in the kernel.

3.4.1 Determinants of Gait Metric

In [24], Saunders *et al.* describe determinants of gait, dealing with conserving energy and maintaining forward momentum during human walking. Using this line of work as inspiration, we based our metric on gait determinants for knee flexion in swing, ankle movement and center of mass trajectory. With this characterization, each point $\boldsymbol{x}$ from the original parameter space can be mapped to a new point $\phi(\boldsymbol{x})$, reflecting determinants of gait measured during the first few steps of using parameters $\boldsymbol{x}$ for control. Consequently, kernel similarities $k(\boldsymbol{x}_i, \boldsymbol{x}_j)$ become a function of the transformed points:

$$k(\boldsymbol{x}_i, \boldsymbol{x}_j) \rightarrow k(\phi(\boldsymbol{x}_i), \phi(\boldsymbol{x}_j))$$

To compute $\phi(\boldsymbol{x})$ we make the following measurements:

1. $M_1 = \theta^{thr}_{high} > \theta^{swing}_{knee} > \theta^{thr}_{low}$, describing whether the knee is flexed enough in swing

2. $M_2 = (\theta_{ankle}^{strike} < 0) \wedge (\theta_{ankle}^{t.o.} > 0)$, noting whether there is heel-strike and toe-off in the gait
3. $M_3 = (Y_{CoM}^{strike} < Y_{CoM}^{midst}) \wedge (Y_{CoM}^{t.o.} < Y_{CoM}^{midst})$, which holds if the center of mass movement is approximately oscillatory between steps
4. $M_4 = \theta_{torso}^{mean} > 0$, which holds if the torso is leaning forward
5. $M_5 = ||v_{avg} - v_{human}||$, expressing the deviation from average human walking speed

In the above, θ_{knee}^{swing} is the knee joint angle in swing, θ_{thr}^{below} and θ_{thr}^{above} are the low and high thresholds on knee angle, similar to human data as described in [32]. Sufficient knee flexion is important for an efficient walking gait, while too much can be energy consuming. θ_{ankle}^{strike} is the ankle joint angle at heel strike (start of stance) and $\theta_{ankle}^{t.o.}$ is the ankle joint angle at take off (end of stance). Respectively, the two conditions ensure heel-strike and toe-off. Y_{CoM}^{strike}, Y_{CoM}^{midst} and $Y_{CoM}^{t.o.}$ are center of mass heights at heel strike, mid stance and takeoff. The condition ensures an oscillatory nature of the CoM movement, which is a natural outcome of human walking. θ_{torso}^{mean} is the mean torso angle and it should be leaning forward for a energy efficient forward movement. v_{avg} and v_{human} are the average simulator speed and the average human walking speed, $1.3m/s$. This term is usually insignificant as compared to the other terms but helps eliminate some special cases, such as the model stepping in place.

A binary score $\in \{0, 1\}$ is given for M_{1-4}^{step} per step, and the final metric is computed as a sum over the steps:

$$score_i^{step} = \Sigma_{k=1}^{5} M_k^{step} \qquad \phi(\boldsymbol{x}) = \Sigma_i score_i^{step}$$

With this we obtain a collapsed 1D representation of our original 16D parameter space. Our metric provides a very coarse indication of whether a particular set of policy parameters would induce stable walking over longer simulation periods. But importantly, if used as a similarity measure, points that lead to very unstable movements appear close together, since they get a score close to zero.

Points that score high on our metric represent policies that can be expected to walk well, but not necessarily optimal for arbitrary cost functions and terrains, since this metric is evaluated over a very short interval in which the model only takes 2-3 steps. This makes the metric good at distinguishing bad points from good points, but not necessarily good points amongst themselves, so further optimization is needed to search for the optimum. Since this metric has no explicit information of the specific cost optimized, or even the optimization method used, it can be used when optimizing with different cost functions, over slightly disturbed models as well as using various optimization methods.

In the next section we present experiments with kernel that utilizes this metric. We obtained very promising results even with this collapsed 1D version of the metric, but of course one could experiment with incorporating additional gait characteristics, or using methods other than binarizing and summing.

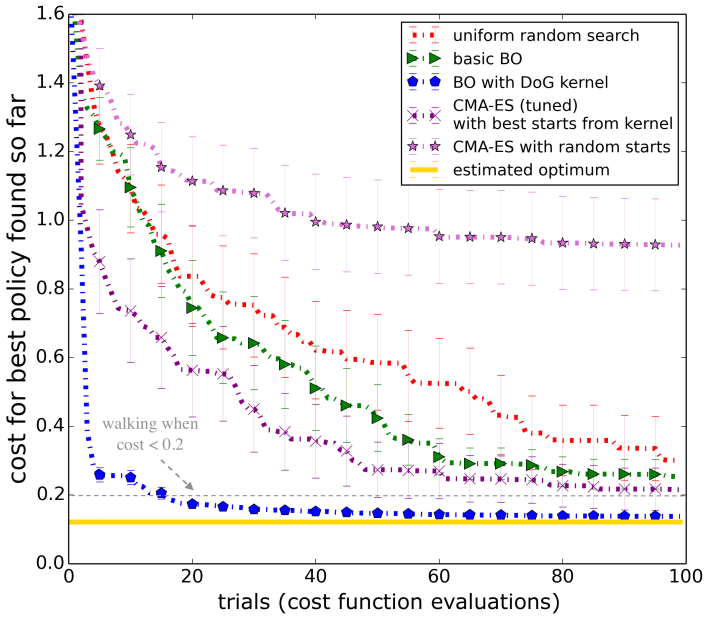
3.4.2 DoG Kernel for Bayesian Optimization

Our capability to detect unpromising policies early in the simulation (described in Section 3.3) made it possible to precompute Determinants of Gait (DoG) scores for a large number of policies. So we generated a sequence of 100K points (a Sobol sequence to maximize coverage of the space without restricting to a fixed grid [3]) and computed DoG scores for these points. We were then able to run Bayesian Optimization with what we call DoG kernel. Instead of Euclidean distances used in the squared exponential kernel (described in Section 2.2), we used the difference in DoG scores: $|\phi(\boldsymbol{x}_i) - \phi(\boldsymbol{x}_j)|$, with $\phi(\cdot)$ as described in Section 3.4.1. We incorporated this kernel into Matlab implementation of Bayesian Optimization from [10]. This implementation had an option to prioritize grid points or points from a pre-computed sequence when considering next candidates, but was not restricted to optimizing only over such points. This allowed us to reuse some of the pre-computed

results to speed up kernel computations when necessary, but did not place restrictions on all optimization experiments. For a further speed up, we used a hashmap to store DoG scores of the points sampled during optimization, since these could be reused in subsequent iterations of BO (e.g. to compute matrix K from Section 2.2).

Figure 4 shows results of our experiments using the DoG kernel. Policies with costs of less than 0.2 corresponded to robot model walking on rough ground, and for BO with DoG kernel ≈ 25 trials were sufficient to find such points. This is in contrast to basic BO that did not get to such results in ≈ 100 trials. To see whether DoG scores could be also used by other algorithms, we ran CMA-ES starting each run from a tight region around one of the best of 100K points for which we precomputed DoG scores (selected randomly from top 100). After tuning the σ parameter of CMA-ES to make it exploit more around the starting point, we were able find policies that resulted in walking on rough ground after 65-70 trials. These results suggest that DoG scores successfully captured useful information about policy space and were able to effectively focus BO (and to some extent CMA-ES as well) on the promising regions of the policy search space.

Figure 4. Experiments with the Determinants of Gait kernel using the smooth cost.

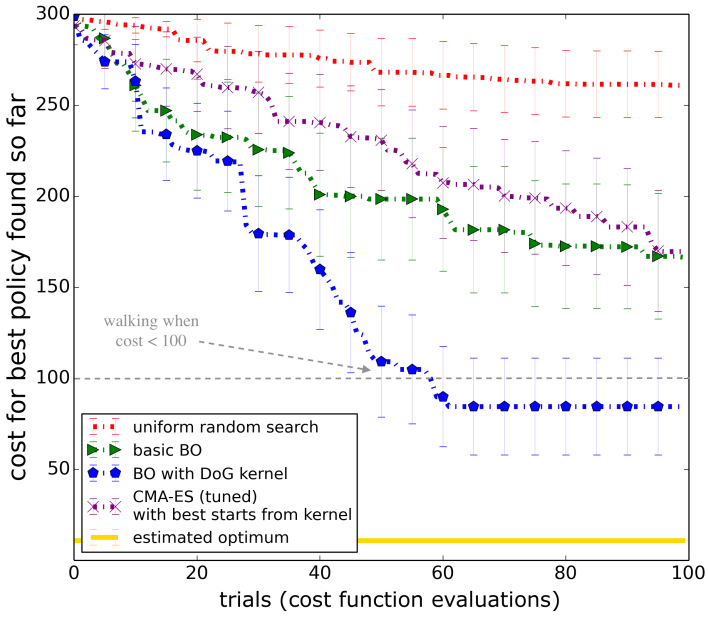


To ensure that our approach can perform well across various cost functions, we also conducted experiments using a non-smooth cost – a slightly modified version of the cost used in [27] for experiments with CMA-ES. This cost function heavily penalizes policies that lead to falls:

$$cost_{CMA} = \begin{cases} 300 - x_{fall} & \text{if fall} \\ 100||v_{avg} - v_{tgt}|| + c_{tr} & \\ & \text{if steady walk} \end{cases}$$

Here x_{fall} is the distance travelled before falling, v_{avg} is the average speed in simulation, v_{tgt} is the target speed and c_{tr} is the cost of transport. We observed good performance on this non-smooth cost function as well (see Figure 5), though it was not as remarkable as with the smooth cost.

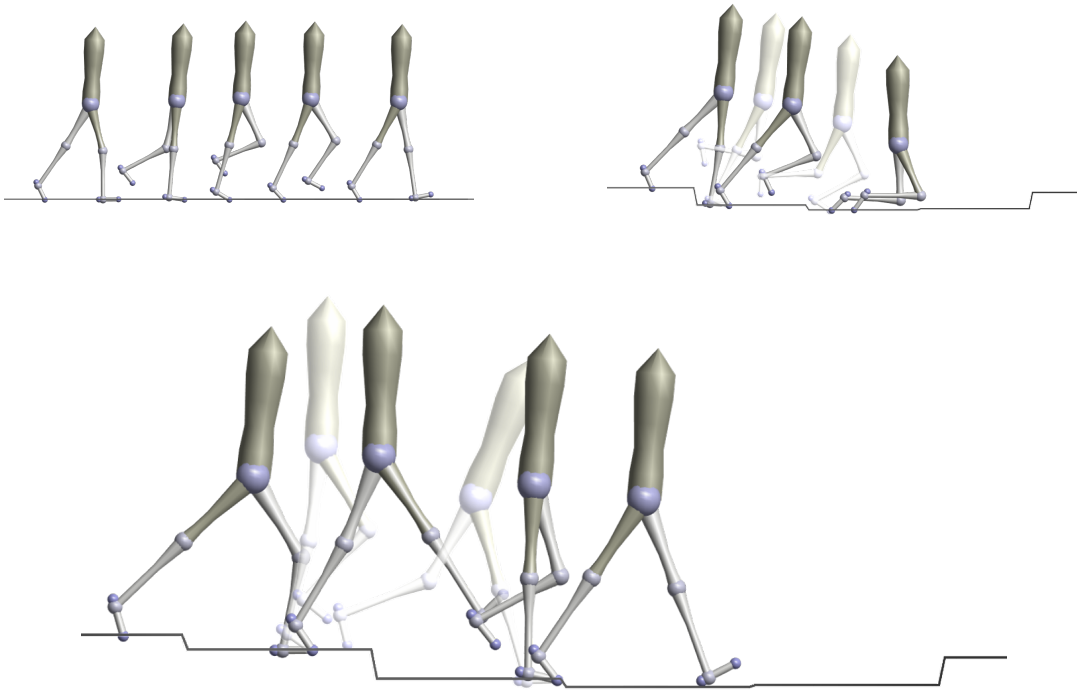
Figure 5. Experiments with the Determinants of Gait kernel using the non-smooth cost.



BO with kernel still outperformed all other methods by a margin, but this different cost seemed to hurt BO and CMA-ES alike. Despite this, BO was able to find successful robust walking in 75% of the optimization runs on rough ground (with height changes of $\pm 6cm$) after only ≈ 60 trials. CMA starting from good kernel points was able to find robust walking policies in only 40% of the runs (after 100 trials).

The above experimental results were very encouraging. Hence we started further experimentation with DoG kernel on other terrain types, like walking up and down a ramp, stairs. Note that we only did the pre-computation for the kernel once – using a pure, unperturbed model of our system on flat ground. Our experiments, however, were conducted on different ground profiles and in the presence of model disturbances (as discussed in Section 3.2). Our perturbed settings were designed such that originally optimal points (set of policy parameters) on flat ground with pure robot model would likely become sub-optimal in the perturbed setting. This is illustrated in the top part of Figure 6, where the policy performing well on flat ground falls on rough ground. This was to ensure the benefit of using the kernel did not come from simple memorization. Points (policy parameters) that had low costs or obtained good DoG scores in the unperturbed system on flat ground were likely to be sub-optimal on other terrains, especially in the presence of non-negligible differences between the simulated model and model with mass and inertial disturbances. Our experiments were set up to test whether our optimization would be able to handle this. And indeed we observed that best policies found in the perturbed setting were able to walk on rough ground (the lower part of Figure 6 presents an example visualization).

Figure 6. Top row: a policy that generates walking on flat ground could fail on rough ground. Bottom row: optimization on rough ground with model disturbances finds policies that succeed, even though pre-computation for the DoG kernel is done using the original model and on flat ground.



The framework we developed for constructing an informed kernel that takes advantage of basic domain knowledge can be used for other domains. This would be possible for the cases where the simulator of the dynamics is available and relevant domain knowledge can be extracted from a large number of short simulations. A domain expert would need to develop a relevant metric (such as the DoG metric we created for locomotion), but this would have to be done only once for a given domain, and then can be used for a range of different cost functions and optimization settings.

4 Optimization for Intelligent Tutoring

In this part we focus on the problem of learning instructional policies that determine the optimal amount of practice to be given to students. We first motivate the approach of using direct policy search and point out the limitations of existing frequently used approaches. Then we note that while Bayesian Optimization could be a good option for direct policy search, it does not exploit all the available structure of the optimal stopping problem. So we develop BO-RESAMPLE: an approach that improves sample efficiency for learning optimal stopping policies. We compare our approach with several state-of-the-art alternatives on simulated student data. Our experiments demonstrate that in the challenging settings of potential model mismatch and lack of prior data our approach outperforms the alternatives. We then discuss experiments with teaching a short course on Histograms, in which we used BO-RESAMPLE to optimize a given instructional objective successfully with only 30 students (participants of the experiment we set up on Amazon Mechanical Turk).

4.1 Model Mismatch and Direct Policy Search

We start by discussing the benefits of direct policy search for complex, hard to model optimal stopping problems. Optimal stopping is concerned with deciding when to stop performing a particular action (giving problems to a student, for example) in order to optimize a certain objective function (e.g. minimize the number of problems given while maximizing the probability the student mastered the material). At each step/iteration an observation of is obtained (e.g. whether the student did the practice exercise correctly) and the decision whether to stop has to be made. If the system decides to stop, then the value of the objective function is computed for the current student (e.g. the assessment score and instruction duration are combined to compute the value). A natural question is whether, rather than optimizing over a parameterized policy class, it is more data efficient to learn a parameterized model class (a model of student behavior/learning, for example) and plan using the resulting model. Indeed, models are well known to often enable more sample efficient reinforcement learning. In addition, there often exist simple policy classes that combine a set of model parameters with additional parameters to create a controller. For example, threshold policies for optimal stopping problems can compare a model-based predictor of the next state with a threshold parameter to decide whether to terminate the process. In the context of a tutoring system, this could correspond to using a statistical model that predicts the probability a student would get a test item correct, with a threshold parameter such that if the probability exceeds the threshold, the system stops teaching that skill. Indeed, such a policy is frequently used in commercial tutoring systems. When a policy can be represented as a combination of a model of the system dynamics plus some additional parameters, it seems potentially beneficial to first learn a set of model parameters, and then optimize over the remaining parameters.

However, most model fitting procedures evaluate parameter quality in terms of predictive power, such as the likelihood of the observed data given an input set of parameters. In contrast, our primary interest is in finding policies that offer good performance with respect to a given objective (cost) function. Particularly when the underlying system may not lie within the class of models considered, it may be neither sufficient nor necessary to find the best fitting model parameters in order to find a well-performing policy. This is important, because in some areas like educational data mining, it is most prevalent to fit a (student) predictive model to the underlying data, and then convert it into an (instructional) policy by combining the learned model with a human-designed or later optimized threshold parameter. We now illustrate how direct policy search can lead to significant benefits in such a setting.

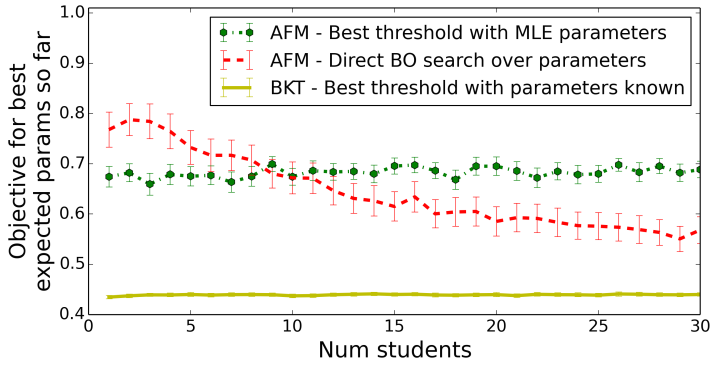
Consider a simple simulation of teaching a student to learn a skill through repeated practice activities. This is an optimal stopping problem because at each time step, after observing whether

a student got the activity correct or not, the agent can decide whether to stop or continue providing the student with additional practice. Upon stopping, the student can do one more practice/question for an assessment. The performance of a policy $\boldsymbol{\pi}$ is defined to be $R(\boldsymbol{\pi}) = I(o_{t+1}=c)\frac{t_{max}-t}{t_{max}+1}$, where I is the indicator function, c stands for correct, and t is number of practice activities completed. Well-performing policies would provide enough practice for a student to do well on the final problem, but avoid over practice (so that a student could be learning new material instead).

To simulate a student we use Bayesian Knowledge Tracing (BKT) models [7] – a modeling approach widely used in the domain of education (the details are described in Section 4.3). To explore what happens when the underlying process does not match the model class used for constructing the stopping policy, we use a threshold policy combined with logistic regression, another very popular student skill learning model. Specifically we use the additive factors model (AFM) introduced in [9] and predict the probability the student will get the next item correct given the history as $p(o_t=c|o_1,\ldots,o_{t-1}) = \frac{1}{1+e^{-(\beta_1+\beta_2 n_c)}}$ where n_c is the number of times the student got the item correct. The instructional policy halts if $p(o_t = c|o_1,\ldots,o_{t-1}) > \beta_3$. The policy class is parameterized by the three parameters $\mathcal{B} = (\beta_1, \beta_2, \beta_3)$.

Figure 7 shows results of comparing direct Bayesian Optimization policy search over the AFM threshold policy to the (often used) alternative of finding the AFM parameters that maximize the likelihood of the observed data, and then fitting a threshold parameter given those parameters. Specifically, we use the MLE parameters to evaluate the possible thresholds according to the objective function (discretizing $[0, 1]$ into 20 discrete thresholds). We average the objective scores for each threshold value, and select the one with the best objective value. The direct BO policy search finds a substantially better policy within only 30 training points. Indeed, we find in an offline brute-force analysis, that the best possible representable AFM threshold policy has an objective value that is tied with the best brute-force fit threshold policy using the true BKT parameters.

Figure 7. Robustness of direct policy search vs. model-fitting plus threshold optimization. BKT parameters: pk=0.18, pl=0.2, pg=0.2, ps=0.1. Objective: minimize $f(\boldsymbol{\pi}) = 1 - R(\boldsymbol{\pi})$ (set up optimization as loss/error minimization).



This simulation encouragingly demonstrates how direct policy search can be robust to mismatch in the assumptions about the underlying generative process, which may be often occur in complex domains that involve user modeling.

4.2 BO-RESAMPLE: Better Sample Efficiency for Optimal Stopping Problems

Policy search may be particularly beneficial in sequential decision making domains when there exists uncertainty about the fitting model class to describe the dynamics of the underlying domain. However, an important limitation of using Bayesian Optimization as a black box optimizer (despite promising prior robotics RL results [20, 6]), is that it leverages none of the underlying structure. Specifically, it ignores the sequential nature of the task: that given a particular set of policy parameters, the resulting policy is used to make decisions of whether to stop or continue across a sequence of observations, and upon the decision to halt, an objective value is obtained. The sequence of observa-

tions, that also deterministically specify the resulting objective function, are not directly used by the Bayesian Optimization process. To our knowledge, [31] was the first work to propose leveraging the structure of sequential decision making processes to accelerate direct policy search with Bayesian Optimization. One of their key methods for doing so is Model-Based Bayesian Optimization Algorithm (MBOA) [31] which builds a (Markov) model of the underlying domain, and uses it to accelerate learning. However, as the choice of models is important and may be wrong, their method weights the model information and may set the weight to zero if the provided information is not helpful. This means that if the provided model is not a useful representation of the domain, MBOA does not gain any benefit over standard black box optimization.

We now propose a simple and intuitive approach for leveraging the structure in optimal stopping domains to increase sample efficiency of BO policy search. Our approach is model-free in the sense that it does not require a model of the dynamics of the underlying domain. It provides a framework for reusing trajectories collected during previous iterations to perform additional off-policy evaluations. "Off-policy" means that we do not need to act in the real environment to make these additional evaluations. Recall that standard BO uses an acquisition function to propose a new input set of (in our case, policy) parameters, and then evaluates the function value of those parameters in the real domain. Our approach instead first does a bounded number of simulated off-policy evaluations, where we use previously collected observation trajectories to simulate the performance of a newly proposed policy. The result of doing so is then passed back to BO as if it was a sample generated in the true domain. Thus we perform this simulated evaluation of other policies in a model-free way. Specifically, a policy that halts at time step t yields a trajectory $(o_1, \ldots, o_{t+1})$, but it also can be viewed as yielding a set of shorter trajectories $(o_1, \ldots, o_{t'})$ where $t' \leq t$. To simulate the execution of a new policy, we simply have to find a prior trajectory that halts at the same or later time step as the new policy, given the same observation sequence. Thus a trajectory can be used to evaluate not just the policy that halted at time step t, but also to provide a sample evaluation for any policy that, under the same history of observations, would have halted at any time step from 1 to t.

Therefore, we can perform off-policy evaluation using the set of trajectories generated from past policy evaluations. We repeat this simulated evaluation process for a fixed number of times, until we have used all prior trajectories at most a bounded number of times or until we reach a point that we cannot evaluate given the existing prior data (if a new policy would halt later than the time step on which all previously evaluated policies have halted, then we cannot evaluate the new policy's simulated performance given prior data). Only then do we take the next proposed policy (set of parameters proposed by BO) and evaluate in the true underlying domain. Pseudocode for this approach is shown in Algorithm 1.

Algorithm 1: BO-RESAMPLE

for $j \leftarrow 1$ **to** J **do**
  $\pi_j \leftarrow$ **NextBOPoint**(GP)
  $\tau_j \leftarrow$ **ExecuteInRealWorld**(π_j)
  UpdateGP(GP, **GetObjectiveVal**(τ_j))
  for $k \leftarrow 1$ **to** K **do** // K is # resamples
    $\pi_k \leftarrow$ **NextBOPoint**(GP)
    // Resample one of previous trajectories
    $\tau_k \leftarrow$ **SelectRandLimited**($\tau_1, ..., \tau_j$)
    $\tau'_k \leftarrow$ **SimulatePolicyExec**(π_k, τ_k)
    UpdateGP(GP, **GetObjectiveVal**(τ'_k))

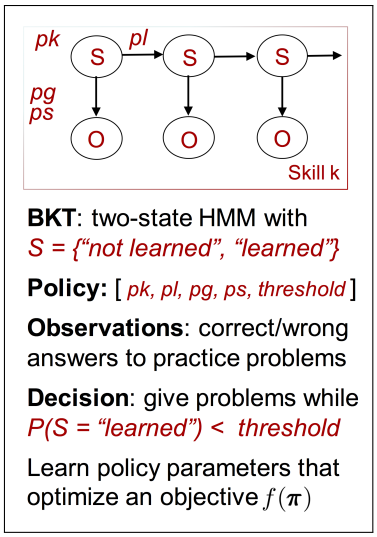
4.3 Student Modeling and Instructional Policies in Education

Education community explored a number of student modeling approaches in the past. One of the most widely used is Bayesian Knowledge Tracing (BKT) [7]. BKT models the process of student answering questions as a two-state Hidden Markov Model: observations are correct/wrong answers, latent state captures whether the student mastered the skill.

BKT yields a 4-parameter model, and the parameters of such model can be learned using a maximum likelihood approach like Expectation Maximization. Figure 8 illustrates BKT model with parameters: *pk* - probability of prior knowledge, *pl* - probability of learning the skill (transitioning from state *"not learned"* to *"learned"*) after seeing another practice/instructional problem, *pg* - probability of guessing the correct answer while being in state *"not learned"*, *ps* - probability of slipping and giving the wrong answer while being in state *"learned"*. "No forgetting" assumption is made: once the student learns the skill, the student does not forget it for the time of instruction (we are only using this to model a single teaching session, not student's knowledge over a school year or a lifetime, so the "no forgetting" assumption is justified in our setting).

Combining BKT with a decision to stop when the probability of student having mastered the skill is above a given threshold generates a 5-parameter instructional policy displayed in Figure 8. BKT and other student modeling approaches have been used in this manner to produce instructional policies. However, most of such prior work focused on maximizing predictive accuracy of the student models and would only add decision-making component in a post-hoc way (for example, heuristically selecting a value for the threshold after the parameters of the model have been fitted). In contrast, in our work we formulate the problem as decision making under uncertainty, conducting direct search for optimal policy parameters (instead of first modeling student behavior and skill inter-dependencies separately, then combining the models into a decision policy by adding decision thresholds afterwards). Additionally, the emphasis in our work is on learning to make decisions in a sample-efficient manner: starting with minimal domain knowledge and finding effective policy parameters within only a small number of interactions with students. With this as a goal, most approaches previously employed in this setting become not applicable due to data scarcity/sparsity.

Figure 8. BKT model and the related instructional policy.

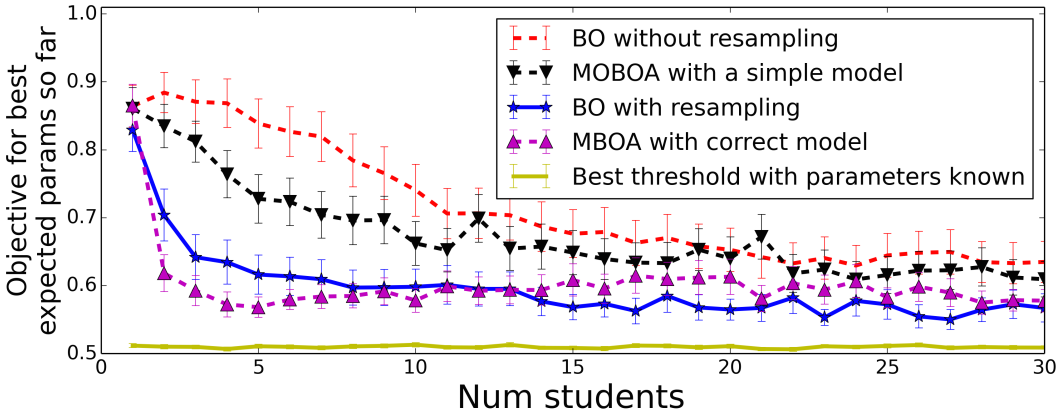


One recent promising direction formulated instructional policy search problem as function optimization, using Bayesian Optimization to optimize a function describing the relationship between instructional policy parameters and student outcomes in a cognitive task [18]. This prior work optimized fixed instructional policies that were independent of individual student performance during instruction (e.g. finding an effective fixed sequence for vocabulary words practice, and using it to teach all students, regardless of progress during practice). We take general inspiration from this approach, but focus on learning adaptive policies, since it is often beneficial to adapt instructional policy based on the learning progress of the student during instruction.

In our setting, observations obtained by the tutoring system consist of interactions with students. A policy $\boldsymbol{\pi}$ mandates giving a certain number of problems (based on the student answers during instruction). At each time step t, the tutoring system receives the current observation o_t (correct/incorrect answer on a practice problem) and must decide whether to halt the instruction for the current student or continue. This decision is made based on the answers to the practice problems the student has given so far. If the tutoring agent decides to halt, it receives one more observation o_{t+1} (result of an assessment) and the instruction for the current student terminates. The policy is evaluated by $f(\boldsymbol{\pi}) = f(o_1, ..., o_{t+1})$, computed according to an objective determined by the system designer.

We also recognize the need to extend standard Bayesian Optimization approaches to allow for better sample efficiency, hence we use our resampling approach (described in Section 4.2) for optimizing optimal stopping instructional policies.

Figure 9. Comparison of BO, BO with resampling and MBOA.
Simulated students using BKT with: pk=0.05, pl=0.05, pg=0.05, ps=0.05.
Objective: minimize $f(\boldsymbol{\pi}) = 1 - R(\boldsymbol{\pi})$ (R defined in Section 4.1).



4.4 Experiments with Simulated Student Data and Model Mismatch

The objective of our experiments described in this section is three-fold: (1) to demonstrate that our resampling approach can improve performance over generic Bayesian optimization policy search for optimal stopping problems, (2) to illustrate the benefit that our model-free approach can have over a previous state-of-the-art model-based method for improving sample efficiency when the choice of the model class is uncertain/poor, and (3) to demonstrate that our proposed method can be used in complex domains where sample efficiency is extremely valuable.

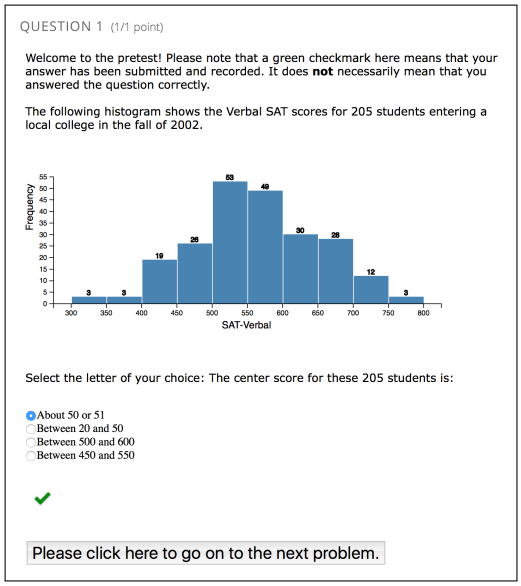
We first consider the simulated student tutor domain with model/policy mismatch: the students were simulated using BKT model, while BO-based algorithms used a policy composed of AFM and a threshold, as described in Section 4.1. We compared our resampling method (using at most 10 resamples of each of the prior data points) to Bayesian Optimization with no resampling. We also compared to MBOA method [31] which fits and leverages a model. MBOA requires an input model class, and we compared: (1) a model class that estimates the probability of correct answer from previously seen trajectories and uses it to predict the performance of a student, and (2) Bayesian Knowledge Tracing. Recall that we use a particular BKT to simulate the student learning process, and so we expect MBOA with a BKT model performs well.

In this experiment we simulated learning trajectories for 30 students. Figure 9 shows the performance of the best policy identified so far. Our resampling approach very quickly finds the parameters of a good policy, substantially faster both than a generic BO approach with no resampling, and than MBOA with a simple model that does not match the underlying student learning process. MBOA with a model that matches the underlying system process does best, as expected. However, as seen, MBOA is sensitive to the model choice, whereas our approach can yield robust benefit in a model-free way. We also examined the cumulative per-step objective for each algorithm which are the actual values experienced (since BO balances exploration and exploitation) and found similar results and ordering as for the best point identified.

4.5 HIT Experiments with Histogram Tutoring System

In order to test the applicability of BO-RESAMPLE to real student behavior, we conducted experiments with a short online course on histograms. This involved creating a course on a custom edX platform and developing back-end and front-end code for running BO-RESAMPLE for policy search. Some aspects of this project have been presented in [2], so here we summarize the setup briefly and

Figure 10. Left: Histogram tutor screenshot. Right: Results of the Amazon turk experiment.



Skill	Give all probl.	BO w/ re-smpl.
d to h	0.37	**0.25**
y axis	0.43	0.46
h to d	0.47	**0.38**
center	0.25	**0.13**
shape	0.65	**0.29**
x axis	0.38	0.36
hist.	0.84	**0.5**
spread	0.08	0.08

Objective values for baseline of giving all problems vs. BO-RESAMPLE minimizing 1-$f(\mathcal{B}_s)$

discuss primarily the aspects specifically related to the performance of our resampling approach.

Histograms are a popular graphical tool to summarize quantitative data and can also be used to introduce more general knowledge about data distributions. Unfortunately, prior research suggests that even after instruction, many students struggle to understand basic properties of histograms [16]. Our histogram tutor consists of a set of multiple choice activities/questions for 8 learning objectives (aka skills), and also includes a fixed assessment of multiple choice questions based on prior literature [16]. Figure 10 displays a screenshot of the tutor. Participants for this experiment were recruited from Amazon turk, they first completed a pre-test (those who did it in less than 2 minutes or got a high score of 8/13 were excluded, since they were assumed to already know the material). After completing practice activities/questions the participants were given post-assessment test to asses their grasp of the material after instruction.

We compared two versions of the tutor. First, we used a baseline of providing all available activities. This represents the approach common in many classrooms where all students receive the same fixed curriculum. Then we used BO-RESAMPLE to learn policies for deciding when to stop providing practice. We maintained one Gaussian Process per skill and used a threshold policy with a BKT model of student learning (as described in Section 4.3). The performance of the policy for a given skill was defined as $f(\mathcal{B}_s) = \frac{p_s + I(p \geq 0.7)}{\sqrt{t_s + 1}}$, where $\mathcal{B}_s$ are the 5 BKT + threshold policy parameters for skill s, p_s is the post-assessment score for skill s, p is the post-assessment score across all skills, and t_s is the number of practice opportunities given on the skill. This objective was designed to identify a policy that enables students to efficiently and effectively learn the desired skills.

In this pilot study we collected data from 19 students/participants for the baseline. We then used BO-RESAMPLE to optimize the policy parameters over 30 students (in a sequential online manner – the usual setting for BO algorithms). We wished to evaluate the effectiveness of the best identified policy parameters after 30 participants, to determine whether our approach had learned a reasonable policy within this number of samples. To do this, we extracted the best policy found after 30 participants, and used this policy for 10 additional participants. Encouragingly, for all but one skill, BO-RESAMPLE found a policy with an average value that met or improved upon the value of the baseline policy (see right side of Figure 10). Though preliminary, these results suggest that BO with resampling may help enable real-world self-optimizing intelligent tutoring systems.

5 Conclusion and Future Work

5.1 Conclusion

In this work we developed approaches to improve sample efficiency of Bayesian Optimization by making use of domain knowledge (simulator of the dynamics) and by using resampling to exploit the structure of optimal stopping tasks to perform model-free off-policy evaluations. We showed how these approaches could be applied to the domains of bipedal locomotion and the task of learning optimal stopping instructional policies in education. We constructed an experimental setup that allowed explicitly testing for robustness to model/policy mismatch and unmodelled/unforeseen disturbances in simulation. This allowed us to anticipate challenging scenarios where the knowledge of the dynamics of the underlying task is imprecise or lacking.

To illustrate how domain knowledge can be effectively incorporated into Bayesian Optimization we presented two approaches to improve sample efficiency when learning bipedal locomotion policies. Our approach of constructing fast priors by using short simulations offered some improvement, however we noted that the improvement would only be substantial in the domains where early termination of simulations is not easy. We then developed the Determinants of Gait metric and the corresponding kernel, which offered significant improvement in sample efficiency of Bayesian Optimization in our experiments in simulation. We pre-computed the kernel on flat ground with unperturbed model, and then tested whether the kernel could offer better sample efficiency in more challenging settings. Optimization with this kernel was able to find effective policies that walked on rough terrain in the presence of model disturbances in only ≈ 25 trials when using a smooth cost (in ≈ 60 when using a non-smooth cost).

To demonstrate the use of task structure for improvements in sample efficiency in a model-free way, we developed a resampling approach for learning optimal stopping policies. We presented experiments with this approach in the domain of education on simulated and real students/participants. These experiments showed that resampling offered improvements over standard Bayesian Optimization in simulation, and was also able to effectively optimize an objective using real interactions with students/participants.

5.2 Future Work

As described in Section 3.2.1, our setup for experiments in locomotion was aiming to approximate the situation where initial pre-computation could be done in a simulated environment, while further optimization to be ultimately done on the real hardware was sample-efficient. The natural next step for this work would be to conduct experiments with full-scale hardware systems. For example, the simulator environment we used could be initialized with the mass and inertial parameters of a humanoid robot. Then, after pre-computing the kernel using simulated dynamics, optimization could be done on the real hardware to learn effective parameters for locomotion control for the robot. We intend to conduct such experiments in the future to investigate the potential of using BO for learning locomotion control for full-scale bipedal robots.

The success of using informed kernel in our locomotion experiments was encouraging. So we also explored the possibility of using a custom kernel for the task of learning optimal stopping policies. In this case it was possible to construct a trajectory-based kernel for deterministic policies. We developed an approach to compute more informed kernel influences using the amount of agreement in actions. While it would be intractable to compute this for all possible trajectories, a useful estimate was to compute the amount of agreement on the states encountered in the trajectories sampled so far.

Starting with a kernel of the form $k(\boldsymbol{p_1}, \boldsymbol{p_2}) = \alpha \cdot \exp(-||\boldsymbol{p_1} - \boldsymbol{p_2}||^2)$, we observe that the quantity $\theta_{\boldsymbol{p_1 p_2}} = \exp(-||\boldsymbol{p_1} - \boldsymbol{p_2}||^2)$ can be thought of as the amount of influence $\boldsymbol{p_1}$ and $\boldsymbol{p_2}$ have on each other. There is, however, some uncertainty about how well the policy parameterization aligns with policy-induced behavior and performance on the objective. So, we can place a Beta prior on $\theta_{\boldsymbol{p_1 p_2}}$, treating the Euclidean distance $||\boldsymbol{p_1} - \boldsymbol{p_2}||^2$ as a prior, and updating $\theta_{\boldsymbol{p_1 p_2}}$ as more sample trajectories become available. Specifically, in our initial implementation we convert initial $\theta_{\boldsymbol{p_1 p_2}}$ to prior agreement counts a and disagreement counts b such that $\frac{a}{a+b} = \exp(-||\boldsymbol{p_1} - \boldsymbol{p_2}||^2)$ and $a+b$ is the maximum trajectory length in a given domain. Then, each time kernel distances $k(\boldsymbol{p_1}, \boldsymbol{p_2})$ need to be estimated during BO, we use trajectories sampled so far to compute updated agreement counts: the number of times the same action was chosen by both policies $\boldsymbol{p_1}, \boldsymbol{p_2}$ for the same state encountered among the sampled trajectories. We then compute the updated posterior $E[\theta_{\boldsymbol{p_1 p_2}}]$. Initial experiments with this approach gave an indication that it could be used to further improve sample-efficiency of BO. A future direction would be to develop this "quickstart" kernel approach further, perhaps combining it with BO-RESAMPLE to create a more sample-efficient framework for solving optimal stopping problems using Bayesian Optimization.

Further directions include developing a framework for optimizing not only the number of trials (policy evaluations), but also establishing an effective tradeoff of computation needed for precomputing priors, kernels or collecting other useful domain knowledge. Also, finding ways to effectively extract domain knowledge from experts (perhaps in a semi-automated way) for informing Bayesian Optimization effectively across a range of different domains and tasks. Another potential for further improvement is combining approaches that use domain knowledge (like an informed kernel) with those exploiting the task structure in a model-free way (as the resampling approach that allowed off-policy evaluations without a relying on a model or a simulator).

References

[1] Chae H. An, Christopher G. Atkeson, and John M. Hollerbach. *Model-based Control of a Robot Manipulator*. MIT Press, Cambridge, MA, USA, 1988.

[2] Rika Antonova, Joe Runde, Min Hyung Lee, and Emma Brunskill. Automatically Learning to Teach to the Learning Objectives. In *Proceedings of the Third (2016) ACM Conference on Learning@ Scale*, pages 317–320. ACM, 2016.

[3] Paul Bratley and Bennett L Fox. Algorithm 659: Implementing Sobol's Quasirandom Sequence Generator. *ACM Transactions on Mathematical Software (TOMS)*, 14(1):88–100, 1988.

[4] Eric Brochu, Vlad M Cora, and Nando De Freitas. A Tutorial on Bayesian Optimization of Expensive Cost Functions, with Application to Active User Modeling and Hierarchical Reinforcement Learning. *arXiv preprint arXiv:1012.2599*, 2010.

[5] Roberto Calandra, Nakul Gopalan, André Seyfarth, Jan Peters, and Marc Peter Deisenroth. Bayesian Gait Optimization for Bipedal Locomotion. In *Learning and Intelligent Optimization*, pages 274–290. Springer, 2014.

[6] Roberto Calandra, André Seyfarth, Jan Peters, and Marc Peter Deisenroth. Bayesian Optimization for Learning Gaits Under Uncertainty. *Annals of Mathematics and Artificial Intelligence*, 76(1-2):5–23, 2016.

[7] Albert T Corbett and John R Anderson. Knowledge Rracing: Modeling the Acquisition of Procedural Knowledge. *User modeling and user-adapted interaction*, 4(4):253–278, 1994.

[8] Ruta Desai and Hartmut Geyer. Robust Swing Leg Placement Under Large Disturbances. In *Robotics and Biomimetics (ROBIO), 2012 IEEE International Conference on*, pages 265–270. IEEE, 2012.

[9] Karen L. Draney, Peter Pirolli, and Mark Wilson. A Measurement Model for Complex Cognitive Skill. 1995.

[10] Jacob R Gardner, Matt J Kusner, Zhixiang Eddie Xu, Kilian Q Weinberger, and John Cunningham. Bayesian Optimization with Inequality Constraints. In *ICML*, pages 937–945, 2014.

[11] Hartmut Geyer. Simulator for Neuromuscular Models for Biped Locomotion. *https://www.cs.cmu.edu/ hgeyer/Software/*.

[12] Hartmut Geyer and Hugh Herr. A Muscle-Reflex Model That Encodes Principles of Legged Mechanics Produces Human Walking Dynamics and Muscle Activities. *IEEE Transactions on neural systems and rehabilitation engineering*, 18(3):263–273, 2010.

[13] Nikolaus Hansen. The CMA Evolution Strategy: A Comparing Review. In *Towards a new evolutionary computation*, pages 75–102. Springer, 2006.

[14] Scott Clark (Yelp Inc). Introducing MOE: Metric Optimization Engine; a new open source machine learning service for optimal experiment design. *http://engineeringblog.yelp.com/2014/07/introducing-moe-metric-optimization-engine-a-new-open-source-machine-learning-service-for-optimal-ex.html*.

[15] Donald R Jones, Matthias Schonlau, and William J Welch. Efficient Global Optimization of Expensive Black-box Functions. *Journal of Global optimization*, 13(4):455–492, 1998.

[16] Jennifer J Kaplan, John G Gabrosek, Phyllis Curtiss, and Chris Malone. Investigating Student Understanding of Histograms. *Journal of Statistics Education,*, 22(2):1–30, 2014.

[17] Andreas Krause, Ajit Singh, and Carlos Guestrin. Near-optimal Sensor Placements in Gaussian Processes: Theory, Efficient Algorithms and Empirical Studies. *The Journal of Machine Learning Research*, 9:235–284, 2008.

[18] Robert V Lindsey, Michael C Mozer, William J Huggins, and Harold Pashler. Optimizing Instructional Policies. In *Advances in Neural Information Processing Systems*, pages 2778–2786, 2013.

[19] Daniel J Lizotte, Tao Wang, Michael H Bowling, and Dale Schuurmans. Automatic Gait Optimization with Gaussian Process Regression. In *IJCAI*, volume 7, pages 944–949, 2007.

[20] Daniel James Lizotte. *Practical Bayesian Optimization*. Phd, University of Alberta, 2008.

[21] Ruben Martinez-Cantin, Nando de Freitas, Eric Brochu, José Castellanos, and Arnaud Doucet. A Bayesian Exploration-Exploitation Approach for Optimal Online Sensing and Planning with a Visually Guided Mobile Robot. *Autonomous Robots*, 27(2):93–103, 2009.

[22] J Mockus, V Tiesis, and A Zilinskas. Toward Global Optimization, volume 2, chapter Bayesian Methods for Seeking the Extremum. 1978.

[23] Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning)*. The MIT Press, 2005.

[24] J. B. dec. M. Saunders, Verne T. Inman, and Howard D. Eberhart. The Major Determinants in Normal and Pathological Gait. *The Journal of Bone & Joint Surgery*, 35(3):543–558, 1953.

[25] André Seyfarth, Hartmut Geyer, and Hugh Herr. Swing-leg Retraction: A Simple Control Model for Stable Running. *Journal of Experimental Biology*, 206(15):2547–2555, 2003.

[26] Bobak Shahriari, Kevin Swersky, Ziyu Wang, Ryan P Adams, and Nando de Freitas. Taking the Human Out of the Loop: A Review of Bayesian Optimization. *Proceedings of the IEEE*, 104(1):148–175, 2016.

[27] Seungmoon Song and Hartmut Geyer. A Neural Circuitry that Emphasizes Spinal Feedback Generates Diverse Behaviours of Human Locomotion. *The Journal of physiology*, 593(16):3493–3511, 2015.

[28] Niranjan Srinivas, Andreas Krause, Sham M Kakade, and Matthias Seeger. Gaussian Process Optimization in the Bandit Setting: No Regret and Experimental Design. *arXiv preprint arXiv:0912.3995*, 2009.

[29] Nitish Thatte and Hartmut Geyer. Toward Balance Recovery with Leg Prostheses Using Neuromuscular Model Control. *IEEE Transactions on Biomedical Engineering*, 63(5):904–913, 2016.

[30] Nicolas Van der Noot, Auke J Ijspeert, and Renaud Ronsse. Biped Gait Controller for Large Speed Variations, Combining Reflexes and a Central Pattern Generator in a Neuromuscular Model. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6267–6274. IEEE, 2015.

[31] Aaron Wilson, Alan Fern, and Prasad Tadepalli. Using Trajectory Data to Improve Bayesian Optimization for Reinforcement Learning. *The Journal of Machine Learning Research*, 15(1):253–282, 2014.

[32] DA Winter and HJ Yack. EMG Profiles During Normal Human Walking: Stride-to-Stride and Inter-subject Variability. *Electroencephalography and clinical neurophysiology*, 67(5):402–411, 1987.

[33] KangKang Yin, Kevin Loken, and Michiel van de Panne. Simbicon: Simple Biped Locomotion Control. In *ACM Transactions on Graphics (TOG)*, volume 26, page 105. ACM, 2007.