Journal of Machine Learning Research 22 (2021) 1-25 Submitted 4/18; Revised 2/19; Published 1/21

# An Empirical Study of Bayesian Optimization: Acquisition Versus Partition

**Erich Merrill** MERRILER@OREGONSTATE.EDU
**Alan Fern** ALAN.FERN@OREGONSTATE.EDU
**Xiaoli Fern** XFERN@OREGONSTATE.EDU
**Nima Dolatnia** DOLATNIN@OREGONSTATE.EDU
*School of EECS, Oregon State University*

**Editor:** Bayesian Optimization Special Issue

## Abstract

Bayesian optimization (BO) is a popular framework for black-box optimization. Two classes of BO approaches have shown promising empirical performance while providing strong theoretical guarantees. The first class optimizes an acquisition function to select points, which is typically computationally expensive and can only be done approximately. The second class of algorithms use systematic space partitioning, which is much cheaper computationally but the selection is typically less informed. This points to a potential trade-off between the computational complexity and empirical performance of these algorithms. The current literature, however, only provides a sparse sampling of empirical comparison points, giving little insight into this trade-off. The primary contribution of this work is to conduct a comprehensive, repeatable evaluation within a common software framework, which we provide as an open-source package. Our results give strong evidence about the relative performance of these methods and reveal a consistent top performer, even when accounting for overall computation time.

**Keywords:** empirical evaluation, global optimization, black box optimization, bayesian optimization

## 1. Introduction

We consider the problem of optimizing an unknown function $f$ by selecting experiments that each specify an input $x$ and return a response $f(x)$. Given an experimental budget, the goal is to select a sequence of experiments in order to find an input that approximately maximizes $f$. An effective approach to this problem is Bayesian optimization (BO) (Brochu et al., 2010), which assumes a Bayesian prior in order to quantify the uncertainty over $f$ via posterior inference. This posterior can then be used to bias the experiment selection in a variety of ways.

Perhaps the most traditional and widely used BO approach is acquisition-based BO (ABO) (Kushner, 1964; Jones, 2001). The key idea is to define an acquisition function in terms of the posterior, which is then optimized at each iteration to select the next experiment. Two commonly used acquisition functions are Expected Improvement (EI) (Mockus, 1994) and Upper Confidence Bound (UCB) (Srinivas et al., 2010), which have both been shown to be practically effective. In addition, UCB has been shown to have probabilistic guarantees on performance under the assumption that the acquisition function can be perfectly optimized at each iteration. However, both UCB and EI are non-convex, making optimization costly and inexact for higher dimensional functions. Thus, in practice, the theoretical results for ABO do not hold and the selection of each experiment can be computationally expensive.

A recent alternative approach to ABO completely avoids the optimization of acquisition functions. These approaches are inspired by the simultaneous optimistic optimization (Munos et al., 2014) (SOO) algorithm, which is a non-Bayesian approach that intelligently partitions the space based on observed experiments to effectively balance exploration and exploitation of the objective. We will refer to SOO and algorithms derived from it as partition-based global optimization (PGO) algorithms. A key feature of SOO is that it provides finite time performance guarantees with minimal assumptions about the objective function's properties. SOO does not, however, exploit the potential benefits of posterior inference. This has led to variants of SOO, such as BaMSOO (Wang et al., 2014) and IMGPO (Kawaguchi et al., 2015), that integrate posterior inference to better direct SOO's exploration. We will refer to these alternative approaches that integrate posterior inference with PGO approaches as partitioning-based BO (PBO) algorithms. Importantly, the PBO algorithms are able to select each experiments using significantly less computation compared to ABO. At the same time, they maintain probabilistic variants of SOO's performance guarantees.

ABO uses more computation per experiment selection than the partition-based approaches, so one might expect ABO to make higher quality decisions and outperform partition-based methods given the same number of experiments. Thus, ABO may have an advantage for application where the number of experiments is fixed and not limited by the runtime of experiment selection. For example, when experiments involve lengthy wet lab trials or expensive simulations, the computation time required for selecting experiments may be negligible. Alternatively, there are applications where the expense of ABO can limit the number of experiments compared to partition-based methods. For example, if experiments correspond to running fast physics simulations, then the higher computational cost of ABO may be a bottleneck that will lead to running fewer experiments in a fixed time horizon and potentially perform worse than partitioning methods.

The above intuitions suggest a potential trade-off between acquisition-based and partition-based methods, however, the literature provides little guidance regarding this trade-off. Most comparisons between ABO, PBO, and PGO have been either indirect or on a sparse set of problems. For example, the PBO algorithm BaMSOO was shown (Wang et al., 2014) to outperform both SOO and selected ABO algorithms. However, this was on a modest number of problems and for Bayesian hyper-parameters that were selected on a per problem basis, which is not always a realistic real-world use scenario. Later, a non-Bayesian PGO algorithm, LOGO (Kawaguchi et al., 2016), was shown to outperform SOO and BaMSOO, which combined with the prior BaMSOO results was taken as evidence for preferring LOGO over ABO approaches. More recent work on IMGPO (Kawaguchi et al., 2015) shows further benefits of PBO over ABO on a small set of problems. However, the ABO implementations used in those evaluations appear to yield inferior performance to the experience of others. If taken at face value, these prior results suggest that PBO approaches dominate ABO approaches despite their much lower computational cost. Yet most applications of BO are still using ABO approaches.

The primary contribution of this paper is to conduct a more thorough evaluation of these approaches with the aim of understanding when and if one should be preferred. We conduct this investigation using a common software framework, which will be publicly available and allow for complete reproducibility. Importantly, we do not introduce new algorithms or variants of existing algorithms in order to avoid the potential appearance of bias in our evaluation. Our results yield fairly consistent observations across a variety of test problems, shedding light on the relative performance of some of key representative acquisition-based and partition-based algorithms.

## 2. Problem Setup and Background

We consider optimizing an unknown function $f : \mathcal{X} \rightarrow \mathbb{R}$ where $\mathcal{X}$ is a compact $d$-dimensional subset of $\mathbb{R}^d$. The black box function $f$ does not necessarily have a closed form but can be evaluated at any point in the domain. Running an experiment $x$ allows us to observe the outcome $y = f(x)$ at some cost. Given a budget that constrains the number of experiments, the goal is to find a input $x$ that approximately maximizes $f$. Without any constraints on $f$, there is no way to guarantee a near optimal value will be found. Thus, prior theoretical and practical work on this black-box optimization problem typically makes some form of smoothness assumption on $f$ (Munos et al., 2014). Under such assumptions it is possible to design more intelligent optimization procedures that prune away and prioritize parts of the input space based on previously observed experiments.

Bayesian optimization (BO) formalizes smoothness via a Bayesian prior over $f$, which is often represented by a Gaussian Process (GP) (Williams and Rasmussen, 2006). In this work, we will focus on BO using GP priors. A GP is a collection of possibly infinite random variables where any subset is multivariate Gaussian distributed. One advantage of using a GP prior over $f$ is that for any experiment $x$ there is a closed form for the mean and standard deviation of its response $f(x)$, conditioned on the previously observed experiments. A GP is completely specified by its mean function, $m(x)$ and its covariance function, $k(x_1, x_2)$. A common choice in the BO literature, which we follow in our experiments, is to select the prior mean to be zero; that is, $m(x) = 0$ for all $x \in \mathcal{X}$. In addition, we will use the squared-exponential kernel with a width hyper-parameter $\theta_i$ for each input dimension, which is a widely used kernel in the BO and more generally GP literature. Given a set of observed experiments, we select the hyper-parameters via automatic relevance determination (SE-ARD) (MacKay, 1998), which optimizes the marginal likelihood with respect to the hyper-parameters.

To evaluate the performance of each algorithm on each objective function, we calculate its regret after $t$ objective observations $r_t = f(x^*) - f(x_t^+)$, where $x^*$ is the optimum point and $x_t^+$ is the best point the algorithm has observed so far after making $t$ observations. Note that regret for ABO methods is frequently calculated by setting $x_t^+$ to the point that maximizes its GP's mean function after making $t$ observations. Since partitioning-based methods cannot provide a similar prediction, to ensure a fair comparison between the different results we exclusively consider points that the algorithms have observed directly when calculating their regret.

## 3. Description of Algorithms

This section describes the algorithms that are included in our empirical study, which covers three algorithmic classes. First, we describe two widely used algorithms from the class of acquisition-based BO (ABO) algorithms. Second, we describe two partitioning-based global optimization (PGO) algorithms, which do not use a Bayesian prior, but have strong theoretical guarantees on regret. Third, we describe two partitioning-based Bayesian optimization (PBO) algorithms, which incorporate a Bayesian prior into the PGO approaches. A comparison of the most relevant properties of all the algorithms is provided in Table 1.

### 3.1 Acquisition-Based BO

Assume we have the observations $D_t = \{x_{1:t}, y_{1:t}\}$ and we are interested in the distribution of the output $y_*$ of a test point (another experiment) $x_*$. Since we have a GP with prior mean zero, the joint distribution of $y_{1:t}$ and $y_*$ can be written as:

$$\begin{bmatrix} y_{1:t} \\ y_* \end{bmatrix} \sim \mathcal{N}\left(0, \begin{bmatrix} K(x_{1:t}, x_{1:t}) & K(x_{1:t}, x_*) \\ K(x_*, x_{1:t}) & K(x_*, x_*) \end{bmatrix}\right)$$

where $K(\cdot,\cdot)$ is the corresponding covariance matrix using the covariance function element-wise. Having the joint distribution, the conditional distribution of $y_*$ given the observed data can be derived as:

$$y_* | D_t, x_* \sim \mathcal{N}\left(\mu(x_*), \sigma^2(x_*)\right)$$
$$\mu(x_*) = K(x_*, x_{1:t})[K(x_{1:t}, x_{1:t})]^{-1} y_{1:t}$$
$$\sigma^2(x_*) = K(x_*, x_*) - K(x_*, x_{1:t})[K(x_{1:t}, x_{1:t})]^{-1} K(x_{1:t}, x_*)$$

ABO uses an acquisition function as a selection heuristic that evaluates each candidate point based on its mean and variance. Acquisition functions are generally designed so that their high values correspond to potentially high values of the objective function. Starting with some initial observed data points, the covariance matrix of the GP is calculated. Using the posterior mean and variance of each candidate data point, the value of the acquisition function can be obtained. The point that maximizes this value is then selected for observation. The experiment and its observed output get added to the data set and this process can be repeated until a specified horizon or budget is exhausted. Since optimizing the hyperparameters of the kernel function helps fit a more accurate GP model to the data, the kernel parameters are tuned periodically during the iterative process. Pseudocode for the ABO algorithm is provided in Algorithm 1.

In this paper, we consider two of the most popular acquisition functions, namely Expected Improvement (EI) (Mockus et al., 1978) and Gaussian Process Upper Confidence Bound (GP-UCB) (Srinivas et al., 2010). At any time step $t+1$ with prior experiments $x_{1:t}$, the acquisition function EI measures the expected improvement with respect to the current best previously observed objective value $f(x_t^+)$, where $x_t^+ = \arg\max_{x_i \in x_{1:t}} f(x_i)$. Under Gaussian Processes, EI can be analytically computed as follows:

$$A_{EI}(x) = (\mu(x) - f(x^+))\Phi(\tfrac{\mu(x)-f(x^+)}{\sigma(x)}) + \sigma(x)\phi(\tfrac{\mu(x)-f(x^+)}{\sigma(x)})$$

where $\mu(x)$ and $\sigma^2(x)$ are the predicted mean and variance of point $x$. And, $\Phi$ and $\phi$ are the CDF and PDF of the standard normal distribution, respectively.

More recently, GP-UCB with provable cumulative regret bounds with high probability was proposed as a BO algorithm (Srinivas et al., 2010). Formally, the algorithm defines the following acquisition function:

$$A_{UCB}(x) = \mu(x) + \beta_t^{1/2}\sigma(x)$$

where $\beta_t$ are appropriate coefficients that balance exploitation against exploration.

The regret bounds provided by Srinivas et al. (2010) depend on the assumptions on the kernel spectrum and their correspondingly defined $\beta_t$. In practice, the researchers choose the form of $\beta_t$ based on their domain knowledge. For instance, although different from $\beta_t$ used in their theoretical results, Kandasamy et al. (2015b) use $\beta_t = 0.2d \log 2t$ in their experiments where $d$ is the number of dimensions. Since we wish to evaluate the performance of these algorithms in a setting where we have no pre-existing domain knowledge that can be used to effectively set these values, we chose to instead select a value for $\beta_t$ that others had reported experimental success with. In this work, we use $\beta_t = 2\log(|\mathcal{X}|t^2\pi^2/6\delta)$ which was found to be effective by Srinivas et al. (2010) with $\delta = 0.1$, where the smaller value of $\delta$ provides a higher probability of achieving the regret bound. The size of $\mathcal{X}$ is obtained assuming each dimension is discretized into 1000 points.

**Algorithm 1** Bayesian optimization Process

```
Input: $D_0$, $N_{up}$    ▷ $D_0$ is the initial data; the hyperparameters get updated every $N_{up}$ iterations
1: for t = 1,2, ... do
2:     if $N_{up}$ is a divisor of $t$ then
3:         Update the kernel hyperparameters
4:     end if
5:     Given $D_{t-1}$, specify the GP
6:     Select $x_t$ by optimizing the acquisition function
         $x_t = \arg\max_{x \in \mathcal{X}} A(x|D_{t-1})$
7:     Observe $y_t$, the output of $x_t$
8:     Augment data $D_t = \{D_{t-1}, (x_t, y_t)\}$
9: end for
```

### 3.2 Partitioning-Based Optimization

Partitioning-based approaches attempt to use an implicit upper bound on the values contained within cells of varying sizes of the objective function's space. Once a procedure has identified the cells with the most promising upper bound, the algorithms can then direct their search focusing on those promising ones by refining them into smaller, hopefully more informative cells. These newly created cells require additional objective observations so that they can be assigned a value representative of the space each cell encloses.

This loop of selecting promising cells and refining them with additional function evaluations is what drives the partitioning algorithms' search, and the predictable behavior of the growth of the partitioning tree is what permits the algorithms their theoretical guarantees. In this section, we consider two PGO algorithms with strong theoretical guarantees, one of which (LOGO) has claimed in prior work (Kawaguchi et al., 2016) to be competitive with ABO.

#### 3.2.1 Simultaneous Optimistic Optimization (SOO)

The simultaneous optimistic optimization algorithm is a partitioning-based global optimization (PGO) algorithm introduced by Munos et al. (2014). Two advantages of SOO over other algorithms are its weak assumptions about the function being optimized and its relative speed. To meet its preconditions SOO only requires that there exists some semi-metric $\ell$ over the objective such that $f(x^*) - f(x) \leq \ell(x^*, x) \forall x$ (where $x^*$ is the optimum point of the objective function) and that the space can be partitioned into cells with monotonically decreasing 'size' according to $\ell$. However, it is not required to know or estimate this metric for the algorithm. Additionally, since SOO is computationally very simple and does not require the optimization of any auxiliary functions to operate, it can consistently select points for observation in a near-zero amount of time when executed on modern computer hardware. A high-level pseudocode description of the algorithm is provided at Algorithm 2.

SOO works by partitioning its objective function's domain using a tree $\tau$. Each node in the tree represents a hyper-rectangle (cell) in the space and is assigned the observed value of the objective function at the center point of the hyper-rectangle. This single objective sample and the size of the cell is used to reason about the potential function values that could be contained within the cell's region. We use $x_{h,i}$ to represent the center point of the $i$th node at depth $h$ in $\tau$, and $g(x_{h,i})$ for its associated observed value from the objective function $f$. In each iteration, SOO selects a set of promising cells for refinement, where each selected cell is partitioned into three equal-sized sub-cells as its children[1]. If a cell has not been refined, it is referred to as a leaf.

Initially, the partition tree contains only a single leaf node that covers the entire input space. The center of the input space is sampled and the resultant function value is assigned to this node. From that point onward, SOO iteratively selects the leaf node at each depth in the tree with the highest upper bound according to the implied semi-metric $\ell$ for further partitioning (refinement), as long as the selected leaf's upper bound is greater than the upper bound of any leaf of larger cell sizes in $\tau$ during that iteration. Importantly, we do not need to explicitly compute the upper bounds in order to select the leaf with the highest upper bound.

Specifically, since all leaves at a given depth of the tree have cells of the same size, finding the leaf with the highest upper bound at a fixed depth $h$ according to $\ell$ is simply a matter of finding the leaf at that depth with the highest center value $g(x_{h,i})$. Furthermore, we know that if any larger-sized leaf has a greater center value than a smaller-sized leaf, the larger one's upper bound according to $\ell$ must be greater due to its larger size and higher known value at its center. Therefore, we can safely ignore the smaller-sized leaf. As such, as we iterate through each depth level of the partitioning tree and select the leaf with the highest center value at each depth, we only need to consider those leaves whose center value

1. Tie breaking is discussed in Section 4.4.

is higher than that of any larger-sized leaves. This will guarantee that the leaf with the maximum upper bound according to $\ell$ is always selected for expansion regardless of the true definition of $\ell$.

SOO takes as parameter a depth-limiting function $h_{\max}(n)$ that defines the maximum depth to which the partitioning tree can be expanded after $n$ cell refinements. In previous work (Munos et al., 2014) (Kawaguchi et al., 2016), this function has always been defined as $h_{\max}(n) = \sqrt{n}$ so we ignore this parameter and assume that this common setting of $h_{\max}$ is a part of SOO itself.

At a high level, the SOO algorithm can be viewed as consisting of two parts: the cell selection process (lines 8-15), and the cell expansion process (lines 16-23). Since the remaining partitioning-based algorithms involve modifying one or both of these processes, we define a simplified version of the algorithm at Algorithm 3, which is used to enable a more clear definition of the derivative algorithms.

**Algorithm 2** Simultaneous Optimistic Optimization

```
1: Initialize $\tau_h = \emptyset \; \forall h \geq 0$        ▷ $\tau_h$ contains the set of leaves at depth $h$
2: $\tau_0 = \{x_{0,0}\}$        ▷ $x_{0,0}$ is the center of the objective function $f$'s domain
3: $g(x_{0,0}) = f(x_{0,0})$
4: $n = 1$
5: loop
6:     $E = \emptyset$
7:     $v_{\max} = -\infty$
8:     for $h = 0 \ldots \min(\text{depth}(\tau), h_{\max}(n))$ do
9:         $(h, i) = \arg\max_{x_{h,j} \in L_h} g(x_{h,j})$, where $L_h$ contains all leaves of $\tau$ at depth $h$
10:        if $g(x_{h,i}) \geq v_{\max}$ then
11:            $E = E \cup \{x_{h,i}\}$
12:            $v_{\max} = g(x_{h,i})$
13:            $n = n + 1$
14:        end if
15:    end for
16:    for each cell $x_{h,i}$ in $E$ do
17:        Subdivide $x_{h,i}$ into its three resultant children cells $x_{h+1,j_1} \ldots x_{h+1,j_3}$
18:        for each child cell $x_{h+1,j}$ do
19:            $g(x_{h+1,j}) = f(x_{h+1,j})$
20:            $\tau_{h+1} = \tau_{h+1} \cup x_{h+1,j}$
21:        end for
22:    end for
23: end loop
24: return $\arg\max_{x_{h,i} \in \tau} g(x_{h,i})$
```

### 3.2.2 Locally Oriented Global Optimization (LOGO)

Locally Oriented Global Optimization (LOGO) (Kawaguchi et al., 2016), as described in Algorithm 4, is a modification to SOO that introduces a local bias parameter $w$ to achieve

**Algorithm 3** Simultaneous Optimistic Optimization (simplified)

```
1: Initialize the partitioning tree $\tau$ with an observation from the center
2: loop
3:     $E = \textsc{SelectCells}_{\text{SOO}}$
4:     $\textsc{ExpandCells}_{\text{SOO}}(E)$
5: end loop
6: return $\arg\max_{x_{h,i} \in \tau} g(x_{h,i})$
```

a finer control of the exploration-exploitation behavior of the algorithm. In particular, instead of selecting the best leaf in the partitioning tree that meets the refinement criteria at *each depth level* for expansion, LOGO uses the same selection process across disjoint sets of $w$ adjacent depth levels. With this approach, when $w$ is set to a high value LOGO will be more inclined to spend its observation budget exploiting a smaller number of the most attractive cells rather than exploring those in nearby depths that are not as attractive, but would have been expanded by SOO. Note that when $w = 1$, the behavior of LOGO and SOO are identical.

LOGO sets the value of $w$ according to the local bias schedule hyperparameter $W$, which must be a list of positive integers and should be monotonically increasing. At the end of each iteration of the algorithm, if the value of the best cell observed during that step is greater than that of the previous step $w$ is set to the next value in $W$. Otherwise, $w$ is set to the previous value in the schedule. The intuition behind this design is that when the algorithm is succeeding (that is, successively observing higher and higher objective values) it should continue exploiting, which is enabled by increasing $w$. When the opposite situation occurs (that is, the algorithm repeatedly fails to find values that improve on the previous step's observations), the algorithm should instead fall back to more exploration-focused behavior to attempt to more quickly locate the next area that may offer further improvement. A fixed-size schedule is used to avoid edge cases where the algorithm would frequently 'get lucky' early on, causing the value of $w$ to consistently increase to the point where the algorithm is then 'stuck' exhibiting exploitative behavior for the remainder of the optimization.

On some objective functions, this adaptive behavior leads to significantly better performance than SOO's static approach without violating any of the original algorithm's performance guarantees.

### 3.2.3 Dividing Rectangles (DIRECT)

DIRECT (Jones et al., 1993) is a popular partitioning-based optimization algorithm which, while very similar in structure to SOO, is meant to improve on Lipschitzian optimization rather than achieve certain theoretical regret bounds. Its primary differences from SOO are modified cell selection conditions, the lack of the concept of a 'depth limit', and a more thorough expansion procedure when expanding cells which are hyper-rectangles for which all edges are of the same size.

**Algorithm 4** Locally Oriented Global Optimization

```
function SelectCells_LOGO
    $E = \emptyset$
    $v_{\max} = -\infty$
    for $k = 0 \ldots \lfloor \min(\text{depth}(\tau), h_{\max}(n))/w \rfloor$ do
        $D = \tau_{kw} \cup \tau_{kw+1} \cup \ldots \tau_{kw+w-1}$
        $i = \arg\max_{i:x_{h,i} \in L_D} g(x_{h,i})$, where $L_D$ contains all the leaves in $D$
        if $g(x_{h,i}) \geq v_{\max}$ then
            $E = E \cup \{x_{h,i}\}$
            $v_{\max} = g(x_{h,i})$
            $n = n + 1$
        end if
    end for
    return $E$
end function
```

We are considering DIRECT in this evaluation primarily as a benchmark for other partitioning-based optimization algorithms due to its ubiquity and efficacy in practice, so we will not provide a more detailed description of the DIRECT algorithm itself.

## 3.3 Partition-Based Bayesian Optimization

While the PGO methods exhibit predictable average-case performance, they clearly have room for improvement. Their treatment of the objective as a true black box results in the PGO algorithms frequently observing regions of the objective that should seem obviously unpromising. However, without any ability to predict the behavior of the objective at a given point, the algorithm's only choice is to spend a function evaluation to prove to itself that the region in question is 'bad' and can be ignored. Even when a poor region is ignored, though, it is still guaranteed to be expanded once it is the only remaining leaf at its depth level, resulting in even more 'unnecessary' function observations in potentially irrelevant areas of the objective.

Here we examine two methods that attempt to improve PGO optimization by incorporating a GP prior into the procedure to use past observations to better direct the expansion and selection of cells.

### 3.3.1 Bayesian Multi-Scale Optimistic Optimization (BaMSOO)

The Bayesian Multi-Scale Optimistic Optimization (Wang et al., 2014) algorithm (Algorithm 5) is a SOO-based algorithm that uses the same selection strategy, but avoids evaluating a point during expansion if the point is deemed unpromising according to the posterior. Specifically, this is done by comparing the upper confidence bound (UCB) derived from the prior at the locations which are about to be observed to the best function value observed in the tree. If the UCB of a cell's center is smaller than the best value observed, we know that the cell is unlikely to contain the optimum. Therefore, Instead of using up a function observation assigning a value to a cell that is unlikely to be further expanded, the cell is assigned the value of the lower confidence bound of its center point according to the prior. This allows the algorithm to continue as expected by effectively 'ignoring' the unpromising cell instead of spending an objective evaluation to assign a value to the probably-unimportant cell.

**Algorithm 5** Bayesian Multi-Scale Optimistic Optimization

```
1:  function ExpandCells_BaMSOO($E$)
2:      $D$ = the observations in $\tau$ not marked as GP-based
3:      for each cell $x_{h,i}$ in $E$ do
4:          Refine $x_{h,i}$ into its three resultant children cells $x_{h+1,j_1} \dots x_{h+1,j_3}$
5:          for each child cell $x_{h+1,j}$ do
6:              if $U(x_{h+1,j}|D) \geq f^*$ then
7:                  $g(x_{h+1,j}) = f(x_{h+1,j})$
8:              else
9:                  $g(x_{h+1,j}) = L(x_{h+1,j}|D)$
10:                 Mark $g(x_{h+1,j})$ as GP-based
11:             end if
12:             $\tau_{h+1} = \tau_{h+1} \cup x_{h+1,j}$
13:         end for
14:     end for
15: end function
```

### 3.3.2 Infinite-Metric GP Optimization (IMGPO)

IMGPO (Algorithm 6) builds on BaMSOO by further taking advantage of the information encoded in the prior by also using it to guide the cell selection process. In addition to the requirement that any cell suitable for refinement must have a value greater than the value of any larger cell, IMGPO also requires that the cell being considered must contain a UCB greater than the value of any smaller (that is, deeper in the tree) cell.

To determine the UCBs that a cell contains, IMGPO builds a subtree within the cell down to a fixed depth using the same node refinement rules. Instead of observing the value of the function at the center of each cell in the subtree, the UCB of each cell's center point is calculated instead. The highest value in this temporary UCB subtree is then considered to be the best UCB contained within the cell in question. With this approach, IMGPO determines whether or not a cell is worth expanding based on the prior information about the upper bound on the value of its potential children, further allowing the algorithm to ignore areas of the objective that seem unpromising.

This change is also the most significant departure from SOO or any of the partitioning-based algorithms. Every other partitioning algorithm's expansion criteria for a leaf is solely based on the properties of the cells above it in the tree. It follows that, for these algorithms, the top-most leaf of the tree *must* be expanded regardless of its value, leading to predictable grid-search-like behavior as the minimum depth of any leaf in the tree grows. Since IMGPO also evaluates cells for refinement based on how they compare to smaller cells, a cell can remain unexpanded while being the only leaf at its depth level. This seemingly minor change prevents IMGPO from exhibiting the grid-search-like behavior that can be seen in

| Algorithm | Sample Selection | Fits GP | Optimizes Acquisition Function | Sample 'depth limit' |
|---|---|---|---|---|
| SOO | Grid-aligned | No | No | Yes |
| LOGO | Grid-aligned | No | No | Yes |
| DIRECT | Grid-aligned | No | No | No |
| BaMSOO | Grid-aligned | Yes | No | Yes |
| IMGPO | Grid-aligned | Yes | No | No |
| BO | Unrestricted | Yes | Yes | N/A |

Table 1: Comparison of several properties shared by the algorithms being compared.

the PGO algorithms' results. Accordingly, the depth limit function $h_{\max}(n)$ that is used in every other SOO-derived algorithm is no longer necessary for IMGPO.

**Algorithm 6** Infinite-Metric GP Optimization

```
1: function SelectCells_IMGPO
2:     $E = \emptyset$
3:     $D$ = the observations in $\tau$ not marked as GP-based
4:     $v_{\max} = -\infty$
5:     for $h = 0 \dots \mathrm{depth}(\tau)$ do
6:         $(h, i) = \arg\max_{x_{h,j} \in L_h} g(x_{h,j})$, where $L_h$ contains all leaves of $\tau$ at depth $h$
7:         if $x_{h,i}$ would be selected by SelectCells$_{\mathrm{SOO}}$ then
8:             Refine $x_{h,i}$ into a subtree $S$
9:             $U^* = \max_{s_{h',j} \in S} U(s_{h',j}|D)$
10:            $f^*$ = the greatest value among all non-GP-based cells at depths $h' > h$
11:            if $U^* \geq f^*$ then
12:                $E = E \cup \{x_{h,i}\}$
13:                $n = n + 1$
14:            end if
15:        end if
16:    end for
17: end function
```

## 4. Experimental Setup

When reviewing the reported results of the algorithms evaluated in this paper, we found that they were rarely evaluated in a true black-box setting. Instead, the algorithms' hyperparameters were tuned after the fact or alternatively the results were presented with non-standard metrics to best demonstrate the strengths of the proposed approach.

These differences in procedure led to confusing and seemingly contradictory results being presented, which spurred the development of this work. Our goal is to directly compare the performance of each algorithm in a setting where they have minimal knowledge of the objective being optimized with the hopes of resolving some of the confusion one could suffer from trying to collectively reason about the previous works' results.

## 4.1 Hyperparameter Selection

To avoid the issue of ex post facto algorithm tuning, we attempted to separate the process of selecting values and settings for any algorithm's hyperparameters from the evaluation itself. To accomplish this, we looked to previous works that had used the algorithms or the papers that had introduced the algorithms themselves and duplicated settings that were suggested or reported as having good empirical performance. To accomplish this, we selected all the hyperparameters for each algorithm before executing them on any objective functions. The hyperparameter values were selected by duplicated them from the paper that had introduced the algorithm or other works that reported good empirical performance with certain settings. The specific values considered for this evaluation are described in the algorithms descriptions in Section 3.

## 4.2 Software Platform

To perform these experiments, we built a custom, extendable C++ framework that executes easily-repeatable evaluations of arbitrary black-box optimization algorithms. The source code is publicly available at `https://github.com/Eiii/opt_cmp`.

Our framework includes C++ implementations of all partitioning-based algorithms (SOO, LOGO, BaMSOO, IMGPO), while a modified[2] version of the bayesopt library (Martinez-Cantin, 2014) provides the implementation of BO that we use in our evaluation. Although the authors of some partitioning-based algorithms provide implementations of their algorithm, we chose to use custom implementations of the algorithms since each derivative PGO and PBO algorithm can be trivially implemented as a minor extension of a simpler partitioning-based algorithm. This choice to define all the partitioning-based algorithms from the same base behavior also prevents one algorithm from incorrectly appearing more or less effective due to differing design decisions or assumptions made by each author of an implementation.

To ensure a fair comparison between the ABO and PBO methods, the PBO implementations repurpose bayesopt's internal GP functionality when they require the use of a GP.

## 4.3 Black Box Functions

We chose to exclusively use synthetic benchmark functions as objectives for this evaluation to ensure that each benchmark could sufficiently evaluate each objective with the time and resources available. The benchmark functions were selected with the goals of including functions with a wide range of quantitative and qualitative properties and including functions that the authors of the partitioning-based methods used to evaluate their algorithms.

The LOGO paper evaluates its algorithm on the sin_2, branin, Rosenbrock, Hartmann3, Hartmann6, and Shekel (with $m = 5, 7, 10$) functions (Kawaguchi et al., 2016) with varying dimensionality and parameterizations when possible. IMGPO and BaMSOO are also both evaluated on a subset of these objectives (Kawaguchi et al., 2015) (Wang et al., 2014), so we chose to include all of them in our experiments to allow for a more direct comparison

2. Modifications were made to expose previously purely internal functions and data structures as required for the BO-aware algorithms to function. The optimization behavior of the library itself is unchanged.

| Function Name | Dimensionality | Description/Category | Evaluated in Previous Works: |
|---|---|---|---|
| Sin | 2 | Simple | SOO, LOGO, IMGPO |
| Branin | 2 | Simple | LOGO, BaMSOO, IMGPO |
| Rastrigin | 2, 4, 6, 10 | Many local maxima | |
| Schwefel | 2, 4, 6, 10 | Many local maxima | |
| Ackley | 2, 4, 6, 10 | Many local maxima | |
| Rosenbrock | 2, 4, 6, 10 | Valley shaped | LOGO, BaMSOO, IMGPO |
| Hartmann | 3, 6 | | LOGO, BaMSOO, IMGPO |
| Shekel ($m = 5, 7, 10$) | 4 | | LOGO, BaMSOO, IMGPO |

Table 2: Summary of the properties of each objective function used.

between results. Additionally, we chose to include the Rastrigin, Schwefel, and Ackley test functions (Molga and Smutnicki, 2005) in our evaluation to provide more variety among the objectives.

The Rastrigin, Schwefel, Ackley, and Rosenbrock objective functions (Molga and Smutnicki, 2005) can be defined with arbitrary dimensionality $D$. Functions with this property are useful in allowing us to better determine the effect that the dimensionality of the objective has on each algorithm's performance. For these objectives, we evaluated each algorithm on each function with $D =$2, 4, 6, and 10.

A summary of the properties of each objective function can be found in Table 2.

## 4.4 Evaluation Process

To evaluate the set of algorithms we ran 70 iterations consisting of executing each optimization algorithm on each function to a horizon of at least 500 samples. For further comparisons, we extended the horizon for SOO, LOGO, DIRECT, and Random to 10,000 samples and configured BaMSOO and IMGPO to run for the average amount of wall-clock time the ABO algorithms expended across all their runs on each objective. These extended horizons were chosen considering the cost to run experiments and the runtime of each algorithm. Each individual evaluation of one algorithm on one objective function was run on one core of a c4.4xlarge Amazon EC2 instance. The decision to run 70 iterations in total was made beforehand by estimating what was feasible within the compute time available for these experiments.

### 4.4.1 Objective Function Randomization

Since SOO and LOGO are deterministic algorithms, it is possible that the algorithm could get unusually lucky or unlucky on some objective functions. This luck could manifest as making a series of observations and resultant partitioning decisions that happen to result in exceptional behavior that is not representative of the algorithm's average-case performance on objective functions with similar properties. Additionally, since any SOO-derived algorithm must first observe the exact center of the hyper-rectangle that defines the objective function's domain, they are all guaranteed to trivially and immediately achieve zero regret on any objective function whose maximum is at its center. Since neither of these difficulties of evaluating SOO-derived algorithms is relevant to the algorithms' performance on real-world problems, we chose to avoid them by randomizing certain properties of the objective functions used in our evaluation.

The cell refinement procedure shared by SOO and its derived algorithms simply splits evenly along the dimension along which the cell has the longest edge. This approach frequently results in ties between dimensions which must be resolved using a tie breaking procedure. The tie breaking procedure used by the partitioning algorithms is simply to assign a fixed priority order to the dimensions and split the higher-priority dimensions first in the case of a tie. It is plausible that this approach could lead to situations where the assignment of this arbitrary ordering could have a significant impact on the algorithms' performance on some objective functions. To mitigate the possibility that this behavior causes certain algorithms to exhibit non-representative behavior, we randomize the tie break dimension ordering at the beginning of each run of each algorithm.

To allow us to evaluate the performance of the partitioning algorithms on objective functions in which the optimum point is in its center, we randomly shrink the bounds of the hyper-rectangle that define the objective's domain such that the optimum point is still guaranteed to be contained within the new bounds.

These methods of randomizing objective functions are applied to every algorithm in a predictable and repeatable way identical to all algorithms so that, for example, one algorithm will not be advantaged by getting 'easier' domains on certain functions than another algorithm.

### 4.4.2 Hyperparameter Settings

To evaluate the algorithms in a true black-box setting, we use one set of hyperparameters for each algorithm across every objective function. To avoid manually tuning the algorithms with the goal of maximizing their performance on our specific problem set, when possible we use the same hyperparameter settings the authors of each method used during their evaluation process.

We use the implementation of BO from the bayesopt (Martinez-Cantin, 2014) library. For our evaluations, we use the squared exponential kernel with automatic relevance detection in which the parameters are estimated using maximum total likelihood. Each ABO run is started with three randomly chosen initial points (which count against the algorithm's budget), and the GP's parameters are re-estimated from the observed data every second ABO iteration.

The EI criteria is parameter-free, but UCB requires us to define a value for $\beta$. This value is frequently tuned per-objective, but we instead set it to $\beta_t = 2\log\left(|D|t^2\pi^2/6\delta\right)$, which Srinivas et al. (2010) found to be effective. In this case, $t$ is the number of function observations made so far and $\delta$ is a constant set to 0.5.

SOO has no parameters to set. LOGO only requires a list of integers to use as the adaptive schedule $W$ for its local bias parameter. In this work, we set $W = 3, 4, 5, 6, 8, 30$ to duplicate the value chosen by Kawaguchi et al. (2016) in their work that introduced the LOGO algorithm.

BaMSOO and IMGPO both require the use of a GP prior to estimate the upper bound on the objective function's value at certain nodes' locations. Using the GP's estimated mean function $\mu$ and standard deviation function $\sigma$, we define the LCB and UCB as $\mu(x) \pm B_N\sigma(x)$.

For BaMSOO we define $B_N = \sqrt{2\log(\pi^2 N^2/6\eta)}$ as suggested by Wang et al. (2014), and for IMGPO we define $B_N = \sqrt{2\log(\pi^2 N^2/12\eta)}$ as suggested by Kawaguchi et al. (2015). In both cases, $N$ is the total number of times the GP was used to evaluate a node, and the constant $\eta$ is set to 0.05 to duplicate the value used in the experimental results included by Kawaguchi et al. (2015) and Wang et al. (2014).

## 5. Empirical Results

In this section we present the results of our experiments using simple regret as our primary performance metric. In particular, after each run of an optimization algorithm, the simple regret is the difference between the best observed outcome and the optimal value. The reported regrets are averages over 70 independent trials.

### 5.1 General Trends of Regret Curves

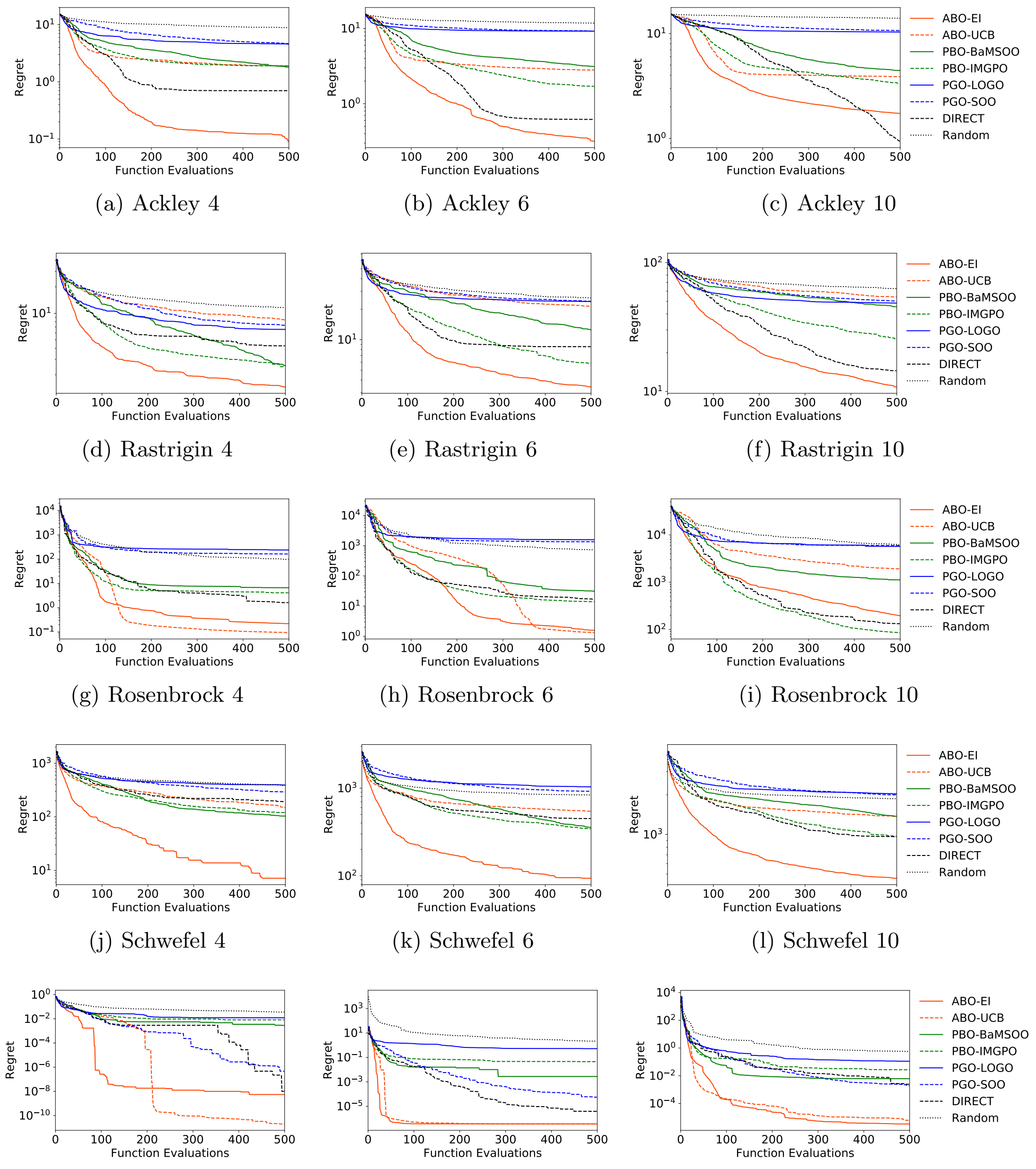
We first consider the performance of the algorithms when given the same experimental budget. Figure 1 shows the average regret curves for each algorithm on the Ackley, Rastrigin, Rosenbrock, and Schwefel functions with dimensionality $D = \{4, 6, 10\}$, as well as the 2D functions Sin2, Branin, and Rosenbrock2. For all but the two-dimensional functions, there appears to be a clear ordering of the different classes of algorithms: ABO-EI overwhelmingly dominates the remaining algorithms at most sample counts with only a few exceptions. The PBO methods are consistently the next closest algorithm class to ABO-EI's performance but are rarely able to achieve the same regret. The PGO algorithms keep up with the best-performing algorithms initially, but their performance quickly seems to 'flat line' and they have difficulty significantly improving further throughout the remainder of the sample budget.

In practice, it is important to understand the worst case performance that we may expect to encounter. Figure 2 shows the worst-case regret for each algorithm observed across the 70 trials at each sample size. Results are shown for three objective functions that are representative of the overall results. Overall, we found that the ordering of the worst-case performance of the algorithms was approximately similar to their relative average-case performances. Most notably, the PGO methods consistently have an inferior worst-case performance than Random on higher dimensional functions.

One difference compared to the average case results is that PBO methods occasionally have significantly poorer worst-case performance than the PGO methods. This is likely due to behavior we have observed in which the GP prior used in PBO methods can be 'tricked' by a few unlucky unappealing function samples near the optimum value that causes the algorithm to ignore what should ideally be seen as a promising region of the function. Since the PGO methods are 'dumber' in that they do not take advantage of a prior and are more likely to fall back to grid-search-like behavior, they do not suffer from this failure case.

### 5.2 Pairwise Comparisons

To gain a better understanding of how the algorithms compare to one another across all the objective functions, we compiled Table 3 which shows a pairwise comparison of all the algorithms across every objective after 500 samples. To generate the table, we calculated the



(a) Ackley 4 (b) Ackley 6 (c) Ackley 10

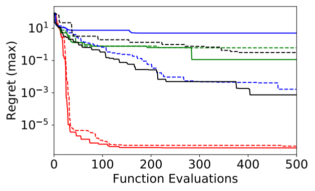
(d) Rastrigin 4 (e) Rastrigin 6 (f) Rastrigin 10

(g) Rosenbrock 4 (h) Rosenbrock 6 (i) Rosenbrock 10

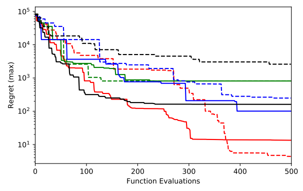
(j) Schwefel 4 (k) Schwefel 6 (l) Schwefel 10

(m) Sin 2 (n) Branin (o) Rosenbrock 2

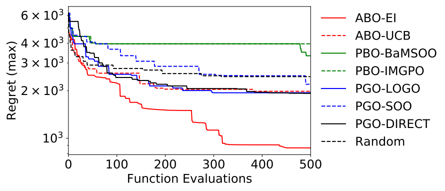
Figure 1: Regret curves for each algorithm on a variety of functions. Each curve shows the regret at each time step averaged across 70 randomized runs. Error bars have been omitted for readability. Statistical significance is considered in Section 5.2.



(a) Branin



(b) Rosenbrock 6



(c) Schwefel 10

Figure 2: Worst-case regret curves for each algorithm. Each curve shows the maximum regret at each time step across 70 randomized runs.

| | ABO-EI | ABO-UCB | PBO-BaMSOO | PBO-IMGPO | PGO-LOGO | PGO-SOO | DIRECT | Random |
|---|---|---|---|---|---|---|---|---|
| ABO-EI | | 12 - 3 - 8 | 12 - 0 - 11 | 10 - 1 - 12 | 18 - 1 - 4 | 18 - 1 - 4 | 3 - 1 - 19 | 23 - 0 - 0 |
| ABO-UCB | 3 - 12 - 8 | | 0 - 2 - 21 | 0 - 4 - 19 | 12 - 1 - 10 | 11 - 1 - 11 | 0 - 6 - 17 | 20 - 0 - 3 |
| PBO-BaMSOO | 0 - 12 - 11 | 2 - 0 - 21 | | 0 - 2 - 21 | 9 - 1 - 13 | 10 - 1 - 12 | 0 - 4 - 19 | 23 - 0 - 0 |
| PBO-IMGPO | 1 - 10 - 12 | 4 - 0 - 19 | 2 - 0 - 21 | | 10 - 1 - 12 | 10 - 1 - 12 | 0 - 2 - 21 | 22 - 0 - 1 |
| PGO-LOGO | 1 - 18 - 4 | 1 - 12 - 10 | 1 - 9 - 13 | 1 - 10 - 12 | | 1 - 2 - 20 | 0 - 13 - 10 | 12 - 1 - 10 |
| PGO-SOO | 1 - 18 - 4 | 1 - 11 - 11 | 1 - 10 - 12 | 1 - 10 - 12 | 2 - 1 - 20 | | 0 - 13 - 10 | 14 - 0 - 9 |
| DIRECT | 1 - 3 - 19 | 6 - 0 - 17 | 4 - 0 - 19 | 2 - 0 - 21 | 13 - 0 - 10 | 13 - 0 - 10 | | 22 - 0 - 1 |
| Random | 0 - 23 - 0 | 0 - 20 - 3 | 0 - 23 - 0 | 0 - 22 - 1 | 1 - 12 - 10 | 0 - 14 - 9 | 0 - 22 - 1 | |

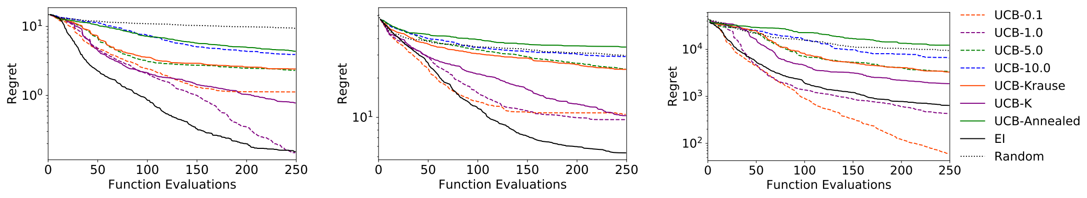
Table 3: Pairwise comparison of each algorithm across all objective functions at $t = 500$ samples. Each cell displays the number of 'wins', 'losses', and 'ties' between the algorithm in the row and the algorithm algorithm in the column (for example, the bottom-left-most cell shows that Random beat ABO-EI 0 times, lost 23 times, and tied 0 times).

95% confidence interval of the mean regret of each algorithm on each objective function at the specified sample size. We considered one algorithm to beat another on a given function if the upper bound of the confidence interval of the mean of its regret was less than the lower bound of the 'challenger' algorithm. If the confidence intervals of the two algorithms' performance overlapped, then they were considered to tie. We use this information to examine differences between different classes of algorithms, across different types of objective functions, and different sample budgets.

**ABO-UCB versus ABO-EI.** ABO-EI and ABO-UCB, despite being similar algorithms, performed very differently from one another compared to other algorithm pairs in the same class. EI wins over UCB over half of the time, and loses only 3 times. While EI consistently beats every other algorithm, UCB seems to be able to at best tie with the PBO algorithms, and could only beat the PGO algorithms approximately half of the time.

Note that this result is based on our selected method for updating the UCB hyperparameter $\beta$. It is important to recall that this choice was based on our best effort to select from the variety of $\beta$ selection methods considered in previous work during a preliminary validation period—we know of no better overall method for $\beta$ selection.

It is very likely that one could tune $\beta$ on a per problem basis to outperform EI, however, this tuning methodology would not result in algorithm behavior that is representative of its efficacy in many real-world scenarios. To test this possibility, we selected a number of additional settings for the $\beta$ parameter and compared UCB's performance using these settings to our previous results and ABO-EI's. Selected results from these experiments are shown in Figure 3.



(a) Ackley 4 (b) Rastrigin 6 (c) Rosenbrock 10

Figure 3: Regret curves for a variety of different $\beta$ values for the UCB algorithm. ABO-EI and Random are provided as benchmarks.

We compared a variety of constant values for $\beta$ (labeled as UCB-x, where x is the constant value), another suggested schedule that was used by Kandasamy et al. (2015a) (labeled as UCB-K), an annealing schedule provided by the Bayesopt library (Martinez-Cantin, 2014) (labeled as UCB-Annealed), the original schedule we used in the previous results (labeled as UCB-Krause), and ABO-EI.

The results show that no one $\beta$ setting appears to be able to match EI's performance more than occasionally. The UCB-K setting does appear to generally do better than the UCB-Krause setting we evaluated, but not to the extent that it would significantly change our results had we selected that schedule instead.

Simple constant values of $\beta$ perform surprisingly well on some objectives, with 0.1 and 1.0 occasionally beating EI. However, the lack of a consistent winner suggests that the $\beta$ value needs to be manually tuned to perform well on each objective function. It's unclear what knowledge about the objective's properties is required to determine which $\beta$ values will be effective or if it's feasible to make that determination during the optimization of an unknown objective function.

On the other hand, EI is parameter free and therefore does not require such a selection process. Thus, these results suggest that ABO-EI is more appropriate to use in a true black-box setting where the objective's properties are unknown, or that further work should be done on determining how to adaptively set ABO-UCB's $\beta$ parameter to improve its black-box performance.

**Acquisition Functions versus Partitioning.** If we consider ABO-EI as the representative of the ABO class of algorithms, it appears that ABO approaches dominate partitioning-based approaches across the board. PBO methods frequently seem to tie with EI's performance, but the fact that they are almost never able to actually outperform EI when given an equal number of objective function observations suggests that EI is the more effective approach in this study.

**PBO versus PGO.** As should be expected, augmenting SOO to enable it to take advantage of Bayesian inference significantly improves the PBO methods' performance over that of the PGO algorithms. While PBO and PGO methods frequently tie with each other in our comparison, both PBO methods only lose once to SOO and LOGO.

**Comparison to Random.** The ABO and PBO algorithms are consistently able to beat Random, with only UCB and IMGPO ever tying with it. PGO methods are not as dominant versus Random, with both SOO and LOGO winning over Random only slightly more often than they're able to tie, and LOGO losing on 1 function.

**Comparison to DIRECT.** DIRECT clearly performs significantly better than either SOO or LOGO in our results. It ties with ABO-EI approximately as often as the other PGO methods lose to ABO-EI, and most notably never loses to either SOO or LOGO while consistently outperforming them.

This is surprising, considering how similar DIRECT is to SOO in its structure and operation. The most notable differences are DIRECT's lack of a 'depth limit' when refining its partitioning tree over the objective's domain, and its lack of a uniform selection of cells to refine among all depths of its partitioning tree.

While these properties are what give SOO its theoretical guarantees, the former may prevent SOO from quickly exploiting an area of the objective that's known to be good while the latter forces it to 'waste' observations exploring areas that might otherwise seem unappealing.

In practice, this appears to suggest that DIRECT's lack of these potential limitations allows it to significantly outperform the PGO algorithms. Since DIRECT's runtime is comparable to SOO and LOGO's, our results do not suggest a potential use case for which SOO or LOGO would be more appropriate to select as an optimization algorithm over DIRECT.

## 5.3 Other Results and Discussion

**Large Numbers of Experiments.** Since PGO algorithms are orders of magnitude faster than ABO and PBO algorithms, and PBO algorithms can be at least an order of magnitude faster than ABO algorithms, it is interesting to consider the performance of each algorithm when the limiting factor is time rather than a sample budget. We present the performance of the algorithms when run for the maximum number of samples considered in this study in Table 4. This simulates the extra objective observations that the faster methods would be able to collect within a fixed time budget, assuming the evaluation of the objective is fast and cheap. These sample sizes were selected based on computational feasibility of conducting the 70 trials for each algorithm on each function.

We allowed the PGO algorithms 10K samples, the ABO algorithms 500 samples, and the PBO algorithms were allowed to run for the same average wall clock run time as the ABO algorithms. This approach resulted in approximately 25k samples on average per problem for BaMSOO, and 4k samples for IMGPO. We empirically observe that both SOO and LOGO improve in comparison to the lower-sample-size ABO and PBO approaches, contrasting the pairwise comparisons in Table 3 and Table 4 reveals that the difference is less significant than would be expected considering the PGO methods are allowed ten to twenty times as many samples as the others. On the other hand, DIRECT seems to be able to take advantage of the extra samples—whereas at 500 samples it was beaten handily by ABO-EI, after 10,000 samples it never loses to ABO-EI and wins against it more than any other algorithm.

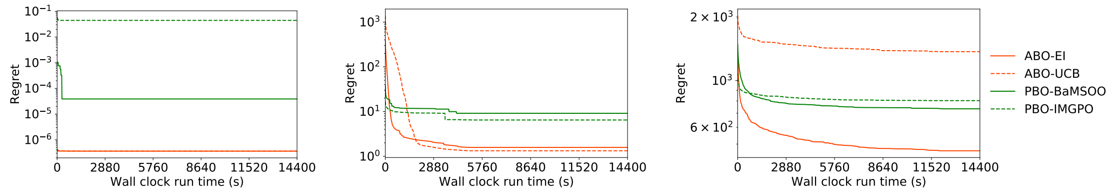
This suggests that the depth-limiting behavior, which is necessary to maintain SOO and LOGO's exploration behavior in earlier stages of the optimization, may be hurting their ability to take advantage of large numbers of objective samples. The resulting coarse grid new samples are restrained to by the limited depth of the refinement tree may be limiting the extent to which the partitioning algorithms are able to quickly 'hone in' on promising regions of the function once they've been identified. Modifying the algorithms' depth limiting $h_{\max}$ function to allow for greater tree expansion at very large number of samples could prevent this behavior, although it may also reduce performance by allowing PGO to spend too many samples exploiting attractive-looking regions earlier on. Investigations during our validation period did not reveal a superior choice for $h_{\max}$ overall.

Comparing PBO versus ABO, we see that both PBO algorithms reduce the number of losses to ABO-EI at the larger sample size, they do not improve the number of wins over ABO-EI. The performance of the PBO algorithms improves slightly against ABO-UCB, however, ABO-UCB was already not very competitive when PBO was allowed only 500 samples.

To more directly compare the performance of PBO and ABO algorithms, we removed the sample budget and applied a time horizon to the PBO methods that allows them to execute for the same average runtime as the ABO method would on the same objective. Due to ABO's need to optimize its increasingly expensive to evaluate acquisition function to select each point, at large numbers of observations ABO runs slowly enough that the PBO methods which are not burdened by this responsibility are able to achieve up to thousands of extra samples of the objective in the same amount of time. We show a sample of representative results obtained with varying time horizon in Figure 4. As our previous results indicate, despite being allowed up to thousands of extra samples of the objective over ABO, the PBO methods are unable to translate those extra samples into a better final regret.

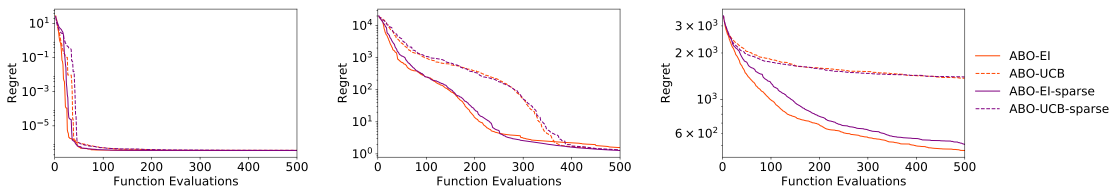
We also evaluated the algorithms' relative performance when the objective function takes more or less time to execute. Our data set contains the wall clock execution time ($e$) and sample count ($t$) for each step in each optimization. Since we know the benchmark functions take near-zero time to evaluate, we assume that the wall clock time represents the amount of time the algorithm itself has consumed up until that point. We can then derive a data set simulating the algorithms' performance on objective functions that takes time $o$ to evaluate by replacing each wall clock time/sample count pair $(e, t)$ in the data set with an updated pair $(e + ot, t)$. We hoped to discover that by adjusting the 'objective complexity' we would could find a trade-off where the rapid sampling pace of the partitioning algorithms would outperform the slower optimization-bound approach of the acquisition algorithms. However, we did not follow through on this analysis once we observed that the partitioning methods were unable to reliably outperform the acquisition methods even when given orders of magnitude more samples.

**'Flexibility' of ABO.** Although we're using one set of hyperparameters on one implementation of ABO for our evaluation, it's important to consider that ABO has many implicit and explicit parameters that can be modified to achieve a desired performance/computation trade off. For example, by tweaking the time allowed by the inner optimization algorithm that optimizes the GP's parameters or the algorithm that optimizes the acquisition function, ABO can be made to run much faster with some unknown penalty to its performance that depends on the objective's sensitivity to the underlying GP's accuracy or the accuracy of the acquisition function optimization. We do not explicitly explore the details of those minor tweaks to ABO in this work. Instead, to examine the impact on ABO's performance



(a) Branin (b) Rosenbrock 6 (c) Schwefel 10

Figure 4: Regret curves for PBO and ABO methods in terms of *wall clock time* rather than number of function evaluations executed. Each curve shows the regret at each time step averaged across 70 randomized runs.



(a) Branin (b) Rosenbrock 6 (c) Schwefel 10

Figure 5: Regret curves for each instance of ABO on a representative selection of functions. Each curve shows the regret at each time step averaged across 70 randomized runs. Error bars have been omitted for readability.

when it is not allowed the same amount of wall clock execution time, we adjust the number of samples in-between GP parameter optimizations.

Figure 5 compares the performance of the instance of ABO we present in this paper to a ‘sparse’ instance that is identical to the implementation of ABO we used other than that it re-learns GP parameters from the observed data one-eighth as frequently. That is, it performs the GP optimization every sixteen observations instead of at every second observation.

Despite an expected decrease in execution time by a factor of eight, the performance of this ‘lighter’ instance of ABO does not appear to perform significantly differently than our ‘standard’ ABO. If ABO-EI did not already appear to be the dominant algorithm in this evaluation, instead presenting a version of it that achieves similar results in one-eighth the time would make it appear even more attractive when compared to the other methods. However, since making this change would not affect our conclusions, we do not consider this ‘lighter’ ABO in our results other than to demonstrate how flexible ABO approaches can be made to be.

**Dependence on Dimensionality.** Because ABO is known to be effective in problems with low dimensionality, we intentionally compared the algorithms on objective functions with a wide range of dimensionality to better understand where and why the partitioning methods’ performance differs from ABO’s. Table 5 shows the same comparison restricted to only two-dimensional objective functions, while Table 6 only shows the results for the objectives with dimension greater than two.

| | ABO-EI | ABO-UCB | PBO-BaMSOO | PBO-IMGPO | PGO-LOGO | PGO-SOO | DIRECT | Random |
|---|---|---|---|---|---|---|---|---|
| ABO-EI | | 12 - 3 - 8 | 15 - 0 - 8 | 6 - 1 - 16 | 13 - 3 - 7 | 13 - 4 - 6 | 0 - 9 - 14 | 23 - 0 - 0 |
| ABO-UCB | 3 - 12 - 8 | | 9 - 0 - 14 | 1 - 6 - 16 | 4 - 2 - 17 | 4 - 2 - 17 | 0 - 12 - 11 | 14 - 2 - 7 |
| PBO-BaMSOO | 0 - 15 - 8 | 0 - 9 - 14 | | 0 - 6 - 17 | 1 - 6 - 16 | 0 - 6 - 17 | 0 - 13 - 10 | 3 - 2 - 18 |
| PBO-IMGPO | 1 - 6 - 16 | 6 - 1 - 16 | 6 - 0 - 17 | | 7 - 0 - 16 | 7 - 0 - 16 | 0 - 2 - 21 | 13 - 0 - 10 |
| PGO-LOGO | 3 - 13 - 7 | 2 - 4 - 17 | 6 - 1 - 16 | 0 - 7 - 16 | | 1 - 0 - 22 | 1 - 10 - 12 | 11 - 3 - 9 |
| PGO-SOO | 4 - 13 - 6 | 2 - 4 - 17 | 6 - 0 - 17 | 0 - 7 - 16 | 0 - 1 - 22 | | 1 - 10 - 12 | 12 - 3 - 8 |
| DIRECT | 9 - 0 - 14 | 12 - 0 - 11 | 13 - 0 - 10 | 2 - 0 - 21 | 10 - 1 - 12 | 10 - 1 - 12 | | 23 - 0 - 0 |
| Random | 0 - 23 - 0 | 2 - 14 - 7 | 2 - 3 - 18 | 0 - 13 - 10 | 3 - 11 - 9 | 3 - 12 - 8 | 0 - 23 - 0 | |

Table 4: Pairwise comparison of each algorithm across all objective functions when algorithms are run for the maximum number of samples considered in this paper. PGO, DIRECT, and Random are allowed $10,000$ samples, ABO is allowed 500 samples, and PBO is allowed the same wall-clock run time as the ABO algorithms.

| | ABO-EI | ABO-UCB | PBO-BaMSOO | PBO-IMGPO | PGO-LOGO | PGO-SOO | DIRECT | Random |
|---|---|---|---|---|---|---|---|---|
| ABO-EI | | 1 - 1 - 4 | 1 - 0 - 5 | 2 - 0 - 4 | 3 - 1 - 2 | 3 - 1 - 2 | 0 - 1 - 5 | 6 - 0 - 0 |
| ABO-UCB | 1 - 1 - 4 | | 0 - 0 - 6 | 0 - 0 - 6 | 2 - 1 - 3 | 3 - 1 - 2 | 0 - 1 - 5 | 6 - 0 - 0 |
| PBO-BaMSOO | 0 - 1 - 5 | 0 - 0 - 6 | | 0 - 0 - 6 | 1 - 1 - 4 | 0 - 1 - 5 | 0 - 1 - 5 | 6 - 0 - 0 |
| PBO-IMGPO | 0 - 2 - 4 | 0 - 0 - 6 | 0 - 0 - 6 | | 1 - 1 - 4 | 0 - 1 - 5 | 0 - 1 - 5 | 5 - 0 - 1 |
| PGO-LOGO | 1 - 3 - 2 | 1 - 2 - 3 | 1 - 1 - 4 | 1 - 1 - 4 | | 1 - 2 - 3 | 0 - 3 - 3 | 4 - 1 - 1 |
| PGO-SOO | 1 - 3 - 2 | 1 - 3 - 2 | 1 - 0 - 5 | 1 - 0 - 5 | 2 - 1 - 3 | | 0 - 3 - 3 | 6 - 0 - 0 |
| DIRECT | 1 - 0 - 5 | 1 - 0 - 5 | 1 - 0 - 5 | 1 - 0 - 5 | 3 - 0 - 3 | 3 - 0 - 3 | | 6 - 0 - 0 |
| Random | 0 - 6 - 0 | 0 - 6 - 0 | 0 - 6 - 0 | 0 - 5 - 1 | 1 - 4 - 1 | 0 - 6 - 0 | 0 - 6 - 0 | |

Table 5: Pairwise comparison of each algorithm across all objective functions with dimension $D = 2$ at $t = 500$ samples.

For the two-dimensional functions we evaluated, it appears that the difference between the algorithms' performance is not as pronounced. While ABO-EI still appears to be the most effective, the PGO approaches frequently tie with its results and even beat it on one occasion each.

For objectives with dimensions greater than two, ABO is much more dominant. ABO-EI only loses to a partitioning method once, and wins against the other algorithms much more frequently than it ties with them. Although we expect ABO's performance to degrade at higher dimensions, it seems that PGO's performance is hit much harder by the increase in objective dimensions. This may be because having more dimensions along which to split the cells means the partitioning tree over the space would be much deeper before PGO can start to refine its search towards promising areas (since each cell must be split along each dimension in some order, regardless of the values observed). This 'refinement' shortcoming, combined with the depth-limiting behavior of PGO algorithms, is likely what causes the performance of PGO algorithms to degrade so consistently on high-dimensional functions.

| | ABO-EI | ABO-UCB | PBO-BaMSOO | PBO-IMGPO | PGO-LOGO | PGO-SOO | DIRECT | Random |
|---|---|---|---|---|---|---|---|---|
| ABO-EI | | 11 - 2 - 4 | 11 - 0 - 6 | 8 - 1 - 8 | 15 - 0 - 2 | 15 - 0 - 2 | 3 - 0 - 14 | 17 - 0 - 0 |
| ABO-UCB | 2 - 11 - 4 | | 0 - 2 - 15 | 0 - 4 - 13 | 10 - 0 - 7 | 8 - 0 - 9 | 0 - 5 - 12 | 14 - 0 - 3 |
| PBO-BaMSOO | 0 - 11 - 6 | 2 - 0 - 15 | | 0 - 2 - 15 | 8 - 0 - 9 | 10 - 0 - 7 | 0 - 3 - 14 | 17 - 0 - 0 |
| PBO-IMGPO | 1 - 8 - 8 | 4 - 0 - 13 | 2 - 0 - 15 | | 9 - 0 - 8 | 10 - 0 - 7 | 0 - 1 - 16 | 17 - 0 - 0 |
| PGO-LOGO | 0 - 15 - 2 | 0 - 10 - 7 | 0 - 8 - 9 | 0 - 9 - 8 | | 0 - 0 - 17 | 0 - 10 - 7 | 8 - 0 - 9 |
| PGO-SOO | 0 - 15 - 2 | 0 - 8 - 9 | 0 - 10 - 7 | 0 - 10 - 7 | 0 - 0 - 17 | | 0 - 10 - 7 | 8 - 0 - 9 |
| DIRECT | 0 - 3 - 14 | 5 - 0 - 12 | 3 - 0 - 14 | 1 - 0 - 16 | 10 - 0 - 7 | 10 - 0 - 7 | | 16 - 0 - 1 |
| Random | 0 - 17 - 0 | 0 - 14 - 3 | 0 - 17 - 0 | 0 - 17 - 0 | 0 - 8 - 9 | 0 - 8 - 9 | 0 - 16 - 1 | |

Table 6: Pairwise comparison of each algorithm across all objective functions with dimension $D > 2$ at $t = 500$ samples.

**Comparison to Previous Results.** Several aspects of our results appear to differ from some of those reported in previous works. The experimental results presented by Wang et al. (2014) suggest that BaMSOO performs better than both SOO and ABO-UCB while being significantly faster than ABO-UCB. Those presented by Kawaguchi et al. (2015) suggest that IMGPO performs better than BaMSOO, SOO, and UCB-EI while being significantly faster than BaMSOO. Later, Kawaguchi et al. (2016) report results that suggests that LOGO is more effective than BaMSOO and significantly more so than SOO.

Taken as a whole, these results seem to suggest that LOGO and IMGPO are the dominant optimization algorithms among those considered, both handily beating the current state-of-the-art ABO methods. Our results contradict this implied ranking. Most notably, we found that ABO-EI was a dominant algorithm and LOGO rarely performs significantly better than SOO.

Of course, we need to be cautious about reasoning about prior results in a transitive fashion. Each evaluation was performed on a different set of black-box functions, sometimes according to different metrics, and likely using different implementations of each algorithm. Still, it's surprising that there is such a discrepancy between our observed relatively performances and those derived from previous work.

For LOGO, the code used to generate prior results was not available due to contractual issues and we have not been able to replicate those results. However, we have validated our implementation with the authors of LOGO. One potential source for the discrepancy is that our evaluation employs randomization of the objective function, which we found was important to avoid observing performance differences due to lucky initializations or grid alignment.

ABO algorithms are necessarily 'tuned' by the authors of papers that use them in comparisons—however, the parameters selected are rarely reported since they're not considered relevant to the paper itself. Although EI is parameter-free, the Gaussian processes used in ABO methods have many parameters that can significantly effect ABO's performance. This may explain the unusually poor performance by ABO-EI in previous work. Without any motivation to maximize the ABO methods' performance in the comparison, it's unclear whether or not the parameters selected for the evaluation result in representative behavior from the algorithm.

## 6. Summary

We presented experimental results comparing PGO, PBO, and ABO methods within a common open-source evaluation framework. The results demonstrate that acquisition-based optimization approaches, specifically ABO-EI, outperform partitioning-based optimization methods when evaluated by the average regret achieved after a given number of function observations in a strict black-box setting. Even when partitioning methods are given significantly more samples of the objective function, they are frequently unable to match the results that ABO-EI can achieve with much fewer samples.

The utility of an extremely computationally cheap black box optimization algorithm is already questionable since the limiting factor in problems that apply these algorithms is usually evaluating the black box function itself rather than the optimization algorithm's runtime. We demonstrate that fast partitioning-based methods tend to 'flat-line' on difficult problems even when given significantly more observations, or at least equal runtime, than competing expensive algorithms. This suggests that there are fewer situations in which PGO optimization should be chosen over BO-enabled methods than one might otherwise assume.

This apparent weakness is described in our results, but not well explained. Follow-up investigations into why or how the partitioning methods fail to improve as consistently as ABO methods can after a large number of samples are warranted. If the algorithm's shortcomings are in fact due to the refinement issue we discuss above, it's possible that modifying the depth limiting function or making other small changes to the algorithm could significantly improve its long-term performance compared to otherwise slower algorithms. Still, the partitioning methods' relative speed and efficiency make them a promising target for future work.

Although UCB is commonly shown to perform well when evaluated on synthetic objective functions, we found that without manually tuning its $\beta$ parameter to maximize its performance it regularly failed to outperform both PBO and PGO methods.

## Acknowledgements

This work was supported by NSF grant IIS-1619433 and DARPA contract N66001-19-2-4035.

## References

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 e-prints*, art. arXiv:1012.2599, Dec 2010.

Donald R Jones. A taxonomy of global optimization methods based on response surfaces. *Journal of global optimization*, 21(4):345–383, 2001.

Donald R Jones, Cary D Perttunen, and Bruce E Stuckman. Lipschitzian optimization without the lipschitz constant. *Journal of Optimization Theory and Applications*, 79(1): 157–181, 1993.

Kirthevasan Kandasamy, Jeff Schneider, and Barnabás Póczos. High dimensional bayesian optimisation and bandits via additive models. In *International Conference on Machine Learning*, pages 295–304, 2015a.

Kirthevasan Kandasamy, Jeff G Schneider, and Barnabás Póczos. High dimensional bayesian optimisation and bandits via additive models. In *ICML*, pages 295–304, 2015b.

Kenji Kawaguchi, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. Bayesian optimization with exponential convergence. In *Advances in Neural Information Processing Systems*, pages 2809–2817, 2015.

Kenji Kawaguchi, Yu Maruyama, and Xiaoyu Zheng. Global continuous optimization with error bound and fast convergence. *Journal of Artificial Intelligence Research*, 56(1):153–195, 2016.

Harold J Kushner. A new method of locating the maximum point of an arbitrary multipeak curve in the presence of noise. *Journal of Basic Engineering*, 86(1):97–106, 1964.

David JC MacKay. Introduction to gaussian processes. *NATO ASI Series F Computer and Systems Sciences*, 168:133–166, 1998.

Ruben Martinez-Cantin. Bayesopt: a bayesian optimization library for nonlinear optimization, experimental design and bandits. *Journal of Machine Learning Research*, 15(1): 3735–3739, 2014.

J. Mockus, V. Tiesis, and A. Zilinskas. The application of bayesian methods for seeking the extremum. In L.C.W.Dixon and G.P. Szego, editors, *Towards Global Optimisation 2*, pages 117–129. 1978.

Jonas Mockus. Application of bayesian approach to numerical methods of global and stochastic optimization. *Journal of Global Optimization*, 4(4):347–365, 1994.

Marcin Molga and Czesław Smutnicki. Test functions for optimization needs. *Test functions for optimization needs*, 101, 2005.

Rémi Munos et al. From bandits to monte-carlo tree search: The optimistic principle applied to optimization and planning. *Foundations and Trends® in Machine Learning*, 7(1):1–129, 2014.

Niranjan Srinivas, Andreas Krause, Sham Kakade, and Matthias Seeger. Gaussian process optimization in the bandit setting: no regret and experimental design. In *Proceedings of the 27th International Conference Machine Learning*, pages 1015–1022. Omnipress, 2010.

Ziyu Wang, Babak Shakibi, Lin Jin, and Nando de Freitas. Bayesian multi-scale optimistic optimization. In *AISTATS*, pages 1005–1014, 2014.

Christopher KI Williams and Carl Edward Rasmussen. *Gaussian processes for machine learning*, volume 2. MIT Press Cambridge, MA, 2006.