Adaptive Experimentation to Find the Best Treatment

Eli Ben-Michael
Department of Statistics, UC Berkeley
ebenmichael@berkeley.edu

Abstract

In order to evaluate the effect of any particular treatment, randomized experimentation is the ideal method. However, we must choose which treatments to evaluate. In many settings the treatments lie in a continuous, multidimensional space, and the goal is to find the treatment with the best average treatment effect. Due to budget, computational, or time constraints, we often aim to find this treatment with as few and as small experiments as possible. We address this problem: given a budget constraint, how do we decide which treatments to give and what experiments to run in order to find the optimal treatment? We compare the problem to that of hyperparameter optimization and adapt successful algorithms from the field to solve the problem. We also propose two novel algorithms which directly incorporate the structure of this problem and empirically compare all of the algorithms. We find that in a variety of settings, our proposed algorithms empirically require an order of magnitude lower budget to achieve the same performance as the best hyperparameter optimization algorithms.

1 Introduction

In the traditional statistical story, we want to evaluate the effect of a specific treatment of some scientific interest. We do a power calculation, decide the number of experimental units to assign the treatment, and run a randomized control trial. However, there are many cases where this story falls apart. Consider the experiments run by the industry which performs more experiments than any other: the technology industry. In those experiments oftentimes there is no particular treatment which we *a priori* want to evaluate the effect of. Instead, we are confronted with a continuous, multidimensional space of possible treatments and *we want to find the best one*. For example, consider the usual use case for A/B testing; a site wants to configure a web page to optimize a metric such as the conversion rate, and runs experiments to evaluate the effect of certain configurations; however, these configurations can incorporate multidimensional choices such as color or position of text. Due to budget, computational, or time constraints, we also want to find the best treatment while running as few and as small experiments as possible. We consider the problem of designing a sequence of experiments which, with a fixed budget on the number of experimental units/total sample size, tries to find the best treatment.

We formalize this as a constrained optimization problem. Given a space of treatments $\mathcal{X}$, which throughout we assume is a box (i.e. $\mathcal{X} = \{x \in \mathbb{R}^d \mid a_1 \leq x_1 \leq b_1, \ldots, a_d \leq x_d \leq b_d\}$), and a mapping $f : \mathcal{X} \to \mathbb{R}$ which maps treatments to their average treatment effect, we want to find an optimal treatment $x^* \in \text{argmin}_{x \in \mathcal{X}} f(x)$. Only noisy zeroth-order black box information is available about the function through queries $f(x) + \epsilon$, where for the purposes of this paper ϵ is i.i.d Gaussian, but in principle could be sub-Gaussian. [1] and [2] consider similar scenarios and assume that the function f is convex. In this setting, they produce a sequence of iterates $\{x_t\}_{t=1}^T$ meant to minimize the *Expected Regret* $\mathbb{E}\left[\frac{1}{T}\sum_{t=1}^T f(x_t) - f(x^*)\right]$. Due to Jensen's inequality, in the convex setting an algorithm which minimizes expected regret minimizes the expected objective error [1, 2]. Our setting is slightly different. First, we do not assume that the function is convex, and second we impose a budget on the number of function queries/samples. This budget corresponds to a budget on the total

number of experimental units allowed over all experiments, and is a natural constraint when each unit costs time and money.

This problem has many parallels to hyperparameter optimization [3, 5]: the problem of tuning the hyperparameters of a machine learning algorithm by minimizing loss on a validation set. In both settings the objective is expensive to compute (either running an experiment or training a model), and only zeroth-order information is available, which in the case of hyperparameter optimization is not noisy. Additionally, the notion of a budget is similar: the budget on the number of units/samples in adaptive experimental design maps on to a budget on the number of iterations or training time for iterative machine learning algorithms (e.g. ones that use gradient descent). However, computational and time costs are distributed differently. In hyperparameter optimization there is a large start-up cost to initializing the machine learning algorithm; it is more time consuming to run through one iteration of stochastic gradient descent on 1000 models than it is to run through 1000 iterations of stochastic gradient descent on one model. In adaptive experimental design there is a similarly high fixed cost for running each experiment; however, within each experiment, each experimental unit can be assigned a different treatment at no additional cost. This implies a practical difference between the two problems: in adaptive experimental design it is possible to take a scatter-shot approach and query the objective in many places with a single sample, while the same is not true for hyperparameter optimization.

In Section 2 we explore two different methods for hyperparameter optimization and adapt them to adaptive experimental design. In Section 3 we propose two new algorithms which take advantage of the scatter-shot approach possible in adaptive experimental design. In Section 4 we empirically compare these algorithms on test cases. In Section 5 we discuss future directions.

2 Hyperparameter Optimization

We consider two diverging methods of hyperparameter optimization, that [3] separates into *configuration selection* and *configuration evaluation*. The former refers to methods which attempt to find good configurations of the hyperparameters quickly, and the latter refers to methods which attempt to allocate more resources to potentially good configurations over potentially bad configurations.

2.1 Configuration Selection with Bayesian Optimization

Algorithm 1 Bayesian Optimization [4]

1: **for** $t = 1, 2, \ldots$ **do**
2: Optimize the acquisition function: $x_t = \text{argmax}_x a(x \mid D_{1:t-1})$
3: Calculate $f(x_t)$
4: Augment the data $D_{1:i} = \{D_{1:t-1}, (x_t, y_t)\}$
5: Update model for f

Bayesian optimization is a tool designed for black box global optimization where function calls are expensive. The idea is straightforward: create a model of the objective function and at each step fit the model to the data and optimize a surrogate function called an *Acquisition Function* to choose the next point at which to query the function(see Algorithm 1). Inherent in this process are two essential choices: the model for the objective and the acquisition function, we now consider the former.

2.1.1 Gaussian Processes for Regression

As seen by the name, Bayesian Optimization incorporates a Bayesian approach to modeling the objective function. As in any Bayesian method, there is a likelihood and a prior, however in this case the prior is over functions. The usual prior is a *Gaussian Process* [4, 5]: which extends a multivariate Gaussian distribution. It is a stochastic process with index set $\mathcal{X}$ for which any finite collection is jointly Gaussian distributed [6]. A Gaussian process can be determined by a mean function $m : \mathcal{X} \to \mathbb{R}$ and a covariance kernel $K : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ and we denote $f \sim \mathcal{GP}(m(x), k(x, x'))$. Typically the mean function $m(x)$ is set to zero and attention is paid to the kernel, as we will see, this still allows for flexible modelling [5]. Letting y denote the observed function value with noise, the

full model is

$$\begin{aligned} f &\sim \mathcal{GP}(m(x), k(x, x')) \\ y_i &\sim \mathcal{N}(f(x_i), \sigma^2) \end{aligned} \tag{1}$$

Defining the matrix $K(X, X)$ such that $[K(X, X)]_{ij} = k(X_i, X_j)$, and marginalizing out f we get that the vector of observations y is distributed

$$y \sim \mathcal{N}(0, K(X, X) + \sigma^2 I) \tag{2}$$

Now to compute the posterior distribution of new sample points $\{(\tilde{x}_i, \tilde{f}_i) \mid i = 1, \ldots, \tilde{n}\}$ we note that the joint distribution is

$$\begin{bmatrix} y \\ \tilde{f} \end{bmatrix} \sim \mathcal{N}\left(0, \begin{bmatrix} K(X, X) + \sigma^2 I & K(X, \tilde{X}) \\ K(\tilde{X}, X) & K(\tilde{X}, \tilde{X}) \end{bmatrix}\right) \tag{3}$$

An exercise in conditioning multivariate Gaussian distributions gives that the posterior is

$$\tilde{f} \mid \tilde{X}, X, f \sim \mathcal{N}(K(\tilde{X}, X)(K(X, X) + \sigma^2 I)^{-1} f, K(\tilde{X}, \tilde{X})(K(X, X)^{-1} + \sigma^2 I)^{-1} K(X, \tilde{X})) \tag{4}$$

Note that even starting with the prior mean function as zero, the posterior mean function is non-zero. In fact, looking at (4) we see that the covariance kernel strongly determines what sort of posterior means are possible. We now turn to these kernels, which corresponds to strong prior assumptions about the function.

One popular choice of covariance kernel is the *Squared Exponential Kernel*

$$k(x, x') = \theta_0 \exp\left(\frac{1}{2} r^2(x, x')\right) \quad r^2(x, x') = \sum_{j=1}^{d} (x_j - x'_j)^2 / \theta_j^2 \tag{5}$$

This kernel is infinitely differentiable, and so corresponds to a prior assumption that the objective is very smooth [6]. A different class of kernels is the Matérn class which provides kernels with different levels of smoothness [6]. [5] argues that the squared exponential kernel is "unrealistically smooth for practical optimization problems", and instead suggest using the *Matérn 5/2 Kernel*

$$k(x, x') = \theta_0 \left(1 + \sqrt{5r^2(x, x')} + \frac{5}{3} r^2(x, x')\right) \exp\left(-\sqrt{5r^2(x, x')}\right) \tag{6}$$

This kernel is only twice differentiable, an assumption which is used for second order optimization methods such as Newton's Method.

Note that these kernels both have $d + 1$ hyperparameters. One approach to set these hyperparameters is to optimize the marginal likelihood of y (see (2)) [6]; another is to place hyper priors on the hyperparameters and approximately integrate out the hyperparameters in the posterior, which can be efficiently done using slice sampling [5, 7].

2.1.2 Acquisition Functions

The final part of Bayesian Optimization is the acquisition function. Let $x^+ = \text{argmax}_{x_i \in x_{1:t}} f(x_i)$ be the current best x, and $\mu(x)$ and $\sigma^2(x)$ be the posterior mean and variance functions of the current Gaussian process f, then three possible acquisition functions are [5]:

- *Probability of Improvement*: We choose the acquisition function to simply be the probability that the function value at x is greater than or equal to the current maximum function value seen.

$$a(x \mid D_{1:t}) = P(f(x) \geq f(x^+)) = \Phi(\gamma(x)) \quad \gamma(x) = \frac{\mu(x) - f(x^+)}{\sigma(x)} \tag{7}$$

- *Expected Improvement*: Since probability of improvement does not take into acount *how much* the function improves, it leads to over-exploitative strategies (see Figure 1 for an example). Instead we can explicitly take this into account and compute

$$\begin{aligned} a(x \mid D_{1:t}) &= \mathbb{E}[\max\{0, f(x) - f(x^+)\} \mid D_{1:t}] \\ &= \sigma(x)(\gamma(x)\Phi(\gamma(x)) + \mathcal{N}(x; \mu(x), \sigma(x))) \end{aligned} \tag{8}$$

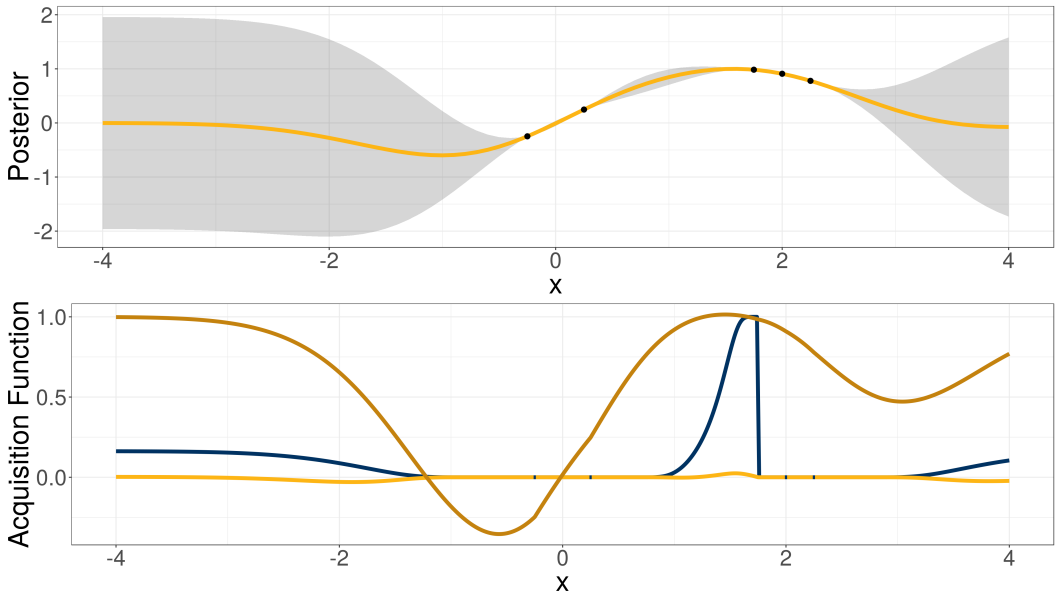


Figure 1: GP posterior along with probability of improvement (blue), expected improvement (gold), and upper confidence bound (brown)

- *GP Upper Confidence Bound*: A different method is to take some κ, which controls exploration vs exploitation, and consider an upper confidence bound on the posterior, a strategy which has been shown to have sub-linear regret [8]

$$a(x \mid D_{1:t}) = \mu(x) + \kappa\sigma(x) \tag{9}$$

See Figure 1 for examples of these three acquisition functions on a synthetic toy example. Note that computing the probability of improvement and the expected improvement require knowing $f(x^+)$. In the noisy case we do not know this value, so [9] suggests a heuristic to replace $f(x^+)$ with $\min_{x \in \mathcal{X}} \mu(x)$.

2.2 Configuration Evaluation with Successive Halving and Hyperband

Bayesian optimization is rather complicated, and requires the practitioner to make several important choices (e.g. the kernel and acquisition functions). Recently, [3] showed empirically that on many hyperparameter optimization tasks, running two instances of random search in parallel is superior to two state-of-the-art variants of Bayesian optimization that they tested. Additionally, random search is guaranteed to converge to the true optimum, no matter the regularity conditions on the objective, while no guarantees exists for state-of-the-art Bayesian optimization algorithms which use various heuristics [3]. This empirical evidence motivates [3] to consider configuration evaluation, and allocate resources in an adaptive manner in order to evaluate many more configuration settings. In the hyperparameter optimization setting the "resources" considered are the number of iterations in iterative algorithms (e.g. the number of gradient descent steps), the size of the training set, or the number of features. Building off of the Sequential Halving algorithm [10, 11], which we consider next, [3] proposed the Hyperband algorithm.

In order to formalize the concept of utilizing resources, [11] consider hyperparameter optimization as an instance of multi-armed bandit best-arm identification. There are two variants of this problem:

- *Stochastic Setting*: In this setting we are given n arms numbered $1, \ldots, n$, where each arm i has a corresponding reward R_i, which is a random variable in $[0, 1]$ with expectation p_i [10, 12]. The arm with the highest expected reward is the *best arm*. At each round we choose an arm to pull, and at the end of the game we must choose the arm we think is best. [10] considers the *fixed confidence* setting, where we must pull the arms as few times as possible to find the best arm with fixed high probability, and the *fixed budget* setting, where we are given a total budget of pulls and we must allocate these pulls so as to find the best

arm with maximal probability. The latter case is of interest to us. Note: it is straightforward to generalize to sub-Gaussian rewards.

- *Non-Stochastic Setting*: In this setting we again have n arms, but rather than associating each arm i with a reward, we associate it with a sequence $\{\ell_{i,t}\}_{t=1}^{\infty}$, where $\ell_{i,t}$ is the reward associated with the t^{th} pull of arm i and we assume that $\nu_i = \lim_{t\to\infty} \ell_{i,t}$ exists. The game remains the same and we want to find the arm i^* with the best ν_i.

For the stochastic, fixed budget setting, [10] proposes Sequential Halving. The algorithm follows a rather simple strategy: run $\log_2 n$ rounds with equal proportions of the budget where at each round we split the per round budget uniformly over all the arms and at the end of the round we examine the empirical average reward for each arm and throw out the worst half (see Algorithm 2). [10] provides bounds on the probability that Sequential Halving fails to correctly identify the best arm. [11] considers Sequential Halving in the non-stochastic setting and provides lower bounds on the budget size required to return the correct algorithm.

Sequential Halving very clearly adapts to hyperparameter optimization: if we take n random hyperparameter configurations, Sequential Halving can perform configuration evaluation and efficiently allocate resources to find the best configuration. However, one question remains: how do we decide how many configurations to take? For large n we do not have many average resources per configuration but get to explore a larger amount of the space, and for small n the opposite is true. Hyperband [3] attempts to address this trade-off between trying many configurations and giving each

Algorithm 2 Sequential Halving [10]

1: **procedure** SEQUENTIALHALVING(budget T)
2: Assign $S_0 = [n]$
3: **for** $r = 0, \ldots, \lceil \log_2 n - 1 \rceil - 1$ **do**
4: Sample each arm $i \in S_r$ for $t_r = \left\lfloor \frac{T}{|S_r| \lceil \log_2 n \rceil} \right\rfloor$

configuration enough resources. The algorithm uses Sequential Halving as a subroutine: it considers different feasible values of n and runs Sequential Halving on n configurations with the same fixed budget (see Algorithm 3). Hyperband requires two choices: R and η, which together define the total budget. If we restrict the total budget then we only have to choose η, high values of which correspond to more aggressive elimination of configurations.

Algorithm 3 Hyperband [3]

1: **procedure** HYPERBAND(R, η)
2: Assign $s_{\text{max}} = \lfloor \log_\eta(R) \rfloor$ and $B = (s_{\text{max}} + 1)R$
3: **for** $s = 0, \ldots, s_{\text{max}}$ **do**
4: Assign $n = \left\lceil \frac{B}{R} \frac{\eta^s}{s+1} \right\rceil$ and $r = R\eta^{-s}$
5: Get n points $T = \{x_1, \ldots, x_n\}$
6: **for** $i = 0, \ldots, s$ **do**
7: Assign $n_i = \lfloor n\eta^{-i} \rfloor$ and $r_i = r\eta^i$
8: For every point in T query the function at every point r_i times
9: Reassign T to be elements of T with the top $\frac{n}{\eta}$ empirical means

2.3 From Hyperparameter Optimization to Finding the Best Treatment

In order to adapt the three hyperparameter optimization algorithms to the problem of finding the best treatment through experimentation, we must define what the experiments are. Both Sequential Halving and Hyperband run a series of rounds with a fixed per-round budget (line 4 in Algorithm 2 and lines 7-9 in Algorithm 3). These rounds naturally map to experiments with fixed sample sizes. For Bayesian optimization we can run an experiment for each call of line 3 in Algorithm 1. So each algorithm has the following interpretation when we fix a total budget for the number of units/sample size:

- *Bayesian Optimization*: Run a series of experiments that give all units the same treatment and adapts the treatment given. The practitioner chooses the sample size of the experiments (or equivalently, the number of experiments to run)

- *Sequential Halving*: Randomly choose treatments and run a series of experiments that adapt the number of units to assign to each treatment. The practitioner must choose the number of treatments.
- *Hyperband*: Run experiments in stages, varying the number of treatments to consider. At each stage adapt the number of units to assign to each treatment. The practitioner must choose how aggressively to eliminate treatments.

3 Two Tree-Based Partition Algorithms

With the hyperparameter optimization algorithms in mind, we propose two algorithms (Algorithms 4 and 5) which attempt to blend configuration selection and evaluation, while taking advantage of the structure of randomized experiments. The idea builds off of Sequential Halving as follows: rather than fixing an initial set of treatments and performing configuration evaluation, we should use information learned about the treatment effect function to also perform configuration selection by replacing discarded treatments with new treatments closer to the remaining treatments. In such a way we can hopefully take advantage of any smoothness in the treatment effect as the treatment varies.

Like Sequential Halving and Hyperband, Sequential Tree (Algorithm 4) proceeds in rounds with equal per-round budgets. In the first round, Sequential Tree takes a scatter-shot approach, querying the function once at uniformly random points (i.e. randomly assigning treatments to units) in order to learn about the objective. Then, it fits a decision tree with m nodes to recursively partition the domain into $\mathcal{P}$. At the end of the first round it restricts $\mathcal{P}$ to the partition elements with predicted function values in the top $\frac{m}{\eta^2}$. In the remaining rounds, for each remaining set in $\mathcal{P}$, the procedure queries the function at uniformly random points inside the set and partitions into into η pieces using a decision tree, restricting to the top $\frac{|\mathcal{P}|}{\eta^2}$ until there is only one set remaining. Essentially, the algorithm partitions the domain into m elements, and then creates a finer and finer partition around points that appear optimal.

Algorithm 4 Sequential Tree

procedure SEQUENTIALTREE(Box constrained space $\mathcal{X} \subset \mathbb{R}^d$, budget T, η, m)
  Assign $\mathcal{P} = \{\mathcal{X}\}$ and `n_nodes` $= m$
  for $r = 1 \dots \lceil \log_\eta(m) \rceil$ **do**
    Assign the number of pulls per element of $\mathcal{P}$, $n_r = \left\lfloor \frac{T}{|\mathcal{P}| \lceil \log_\eta(m) \rceil} \right\rfloor$
    Assign $\mathcal{A}_r = \emptyset$ and $\hat{B}_r = \emptyset$
    for $p \in \mathcal{P}$ **do**
      Remove p from $\mathcal{P}$
      Sample $\{x_{p1}, \dots, x_{pn_r}\}$ independently and uniformly over p
      Let $\{y_{p1}, \dots, y_{pn_r}\}$ be the result of querying the function at these points
      Train a decision tree with `n_nodes` nodes, $t_p = \text{DECISIONTREE}(y_p \sim X_p)$
      Add the `n_nodes` elements of the partition of p defined by t_p to $\mathcal{A}_r$
      Add the prediction from t_p for each element to $\hat{B}_r$.
    if $|\mathcal{P}| > \eta^2$ **then**
      Set $\mathcal{P}$ to be the elements of $\mathcal{A}_r$ with predictions in the top $\frac{|\mathcal{P}|}{\eta^2}$ of $\hat{B}_r$
    else
      Let $\hat{p}$ be the element of $\mathcal{P}$ with the best prediction
      return $\text{MIDPOINT}(\hat{p})$
    `n_nodes` $= \eta$

In Algorithm 4 the practitioner must choose the number of elements in the initial partition, m. In the same vein as the choice of the number of treatments in Sequential Tree, this choice corresponds to a trade off between the amount we can explore the space by making finer partitions, and the amount of queries we have in any particular partition element. As we show empirically in Section 3.1, this choice can have a large effect on the error measured in terms of the objective. Furthermore, if m is set too large then the initial decision tree can over-fit, and areas of the domain which the algorithm suggests are good could be the result of noise.

To address this, Algorithm 5 makes a minor tweak to Algorithm 4: instead of initially partitioning the space into m pieces it partitions into η^2. Because $\eta^2 << m$, this leads to a much coarser initial partition and thus one that is less prone to over-fitting. Additionally, rather than use the decision tree's prediction to decide which sets in $\mathcal{P}$ to keep, the algorithm trains and uses a separate random forest

with k trees to help combat over-fitting (there is no particular reason for a random forest, other than making the code simpler). We now turn to examining how the parameter choices affect performance.

Algorithm 5 Partition Tree

procedure PARTITIONTREE(Box constrained space $\mathcal{X} \subset \mathbb{R}^d$, budget T, η, R, k)
  Assign $\mathcal{P} = \{\mathcal{X}\}$, and $m = \eta^2$
  for $r = 1, \ldots, R$ **do**
    The number of pulls per element of $\mathcal{P}$ $n_r = \left\lfloor \frac{T}{|\mathcal{P}|R} \right\rfloor$
    Assign $\mathcal{A}_r = \emptyset$ and $\hat{B}_r = \emptyset$
    for $p \in \mathcal{P}$ **do**
      Remove p from $\mathcal{P}$
      Sample $\{x_{p1}, \ldots, x_{pn_r}\}$ independently and uniformly over p
      Let $\{y_{p1}, \ldots, y_{pn_r}\}$ be the result of querying the function at these points
      Train a decision tree with m nodes, $t_p = \text{DECISIONTREE}(y_p \sim X_p)$
      Train a random forest with k trees and m nodes per tree
      $rf_p = \text{RANDOMFOREST}(y_p \sim X_p)$
      Add the m elements of the partition of p defined by t_p to $\mathcal{A}_r$
      Add the prediction from rf_p for the mid point of each element to $\hat{B}_r$.
    Set $\mathcal{P}$ to be the elements of $\mathcal{A}_r$ with predictions in the top $\frac{|\mathcal{P}|}{\eta}$ of $\hat{B}_r$
    Let $\hat{p}$ be the element of $\mathcal{P}$ with the best prediction
    $m = \eta$
  return MIDPOINT($\hat{p}$)

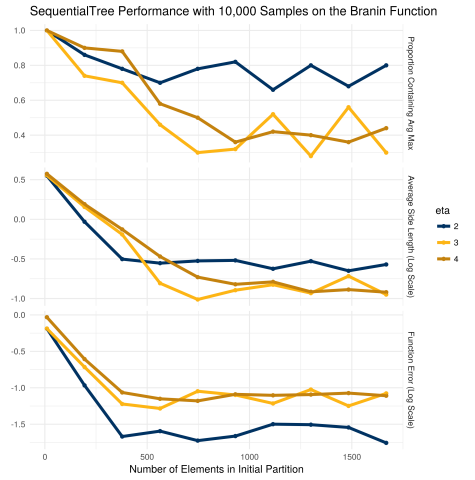
3.1 Choosing Parameter Settings

As discussed in the previous section, for Algorithm 4 the practitioner needs to choose m, the size of the initial partition. For Algorithm 5, there is a choice of the number of rounds R, and the number of trees in the random forest k. In both Algorithms we must choose η which, as in Hyperband, controls how aggressively the algorithm discards configurations. The number of trees k in Algorithm 5 could ostensibly be set as large as the practitioner wants, and in our evaluations we set it to 10. In order to evaluate the effects of the other parameters we run 50 trials where we fix a budget of 50,000 samples and optimize the Branin function, a 2-dimensional function with 3 global minima that is used as a benchmark for Bayesian optimization[1] [5, 13].

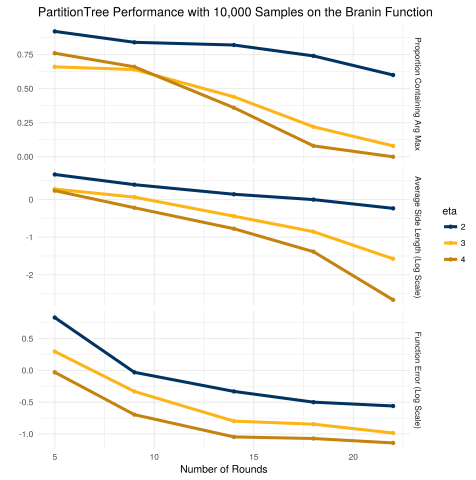
Figures 2 and 3 show the results of these trials. We consider a number of criteria to evaluate the algorithms. First we consider the proportion of times that a true maximizer of the objective was contained inside of the final set, and the size of the maximum side length of that set. Since both algorithms return the midpoint of the set, these two values give an empirical indication of the probability that a true maximizer is contained in a $\|\cdot\|_\infty$ ball around the output of the algorithm. The top two panels of Figure 2 (a) and (b) show these two values for Sequential Tree and Partition Tree. We see that for $\eta = 2$, the proportion of final sets which contain a true maximizer remains around 75% for Sequential Tree, although this corresponds with a larger final $\|\cdot\|_\infty$ ball. It is unclear if these values even matter, so we also consider the error as measured by the difference between the true max and the output of the algorithm, seen in the bottom panel of Figure 2 (a) and (b). Here we see two parallel and diverging trends between Algorithms 4 and 5. As we increase the number of elements in the initial partition or the number of rounds, which both create finer partitions, there are diminishing returns, and the error flattens out. However, Algorithm 4 appears to perform better with a less aggressive setting of η, while Algorithm 5 benefits from a higher η.

We inspect this difference by considering the difference between the true max and the midpoint of the set each algorithm predicts is the best at each round. Looking at Figure 3 (a) we see that for $\eta = 2, 3, 4$, at each round Sequential Tree performs about the same, but $\eta = 2$ does the most number of rounds. Perhaps this shows that being aggressive does not help at each round, so the least aggressive setting wins out because it does more rounds. In contrast, Figure 3 (b) shows that more aggressive settings of η in Partition Tree actually result in lower error at each round. Since in this algorithm the choice of η does not dictate the number of rounds, larger settings clearly win out. For future work it would be interesting to see if a tweak to Partition Tree where η starts aggressive but then becomes less aggressive would work. Perhaps this would lead the algorithm to get past the apparent error floor seen in Figure 3 (b).

[1] Details, and implementation, and a visualization of this function at `www.sfu.ca/ ssurjano/branin.html`

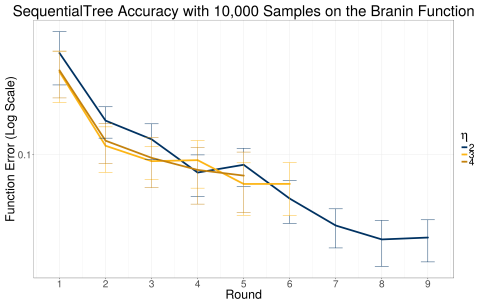


(a) The choice of η and the number of elements in the initial partition can have a large effect on the performance of Algorithm 4

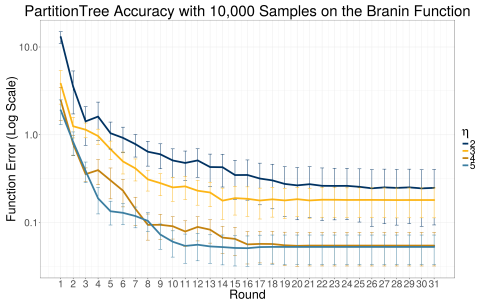


(b) Increasing the number of rounds has a diminishing effect on the performance of Algorithm 5, and more aggressive settings of η appear to perform better

Figure 2: For different values of the size of the initial partition in Sequential Tree (a) and number of rounds in Partition Tree with $k = 10$, (b), and different values of η, 50 trials were run on the Branin function with a budget of 10,000 samples.



(a) With $\eta = 2$ Sequential Tree is less aggressive and explores more of the space. Empirically this gives better performance.



(b) For all settings of η, Partition Tree reaches an error floor with successive rounds.

Figure 3: We run the same trial as in Figure 2 and inspect how the error in the objective behaves after each round

4 Empirical Evaluation

We evaluate and compare Algorithms 1-5 by running trials on several functions and at different noise levels. The functions that we consider are the Branin, Hartmann 3d and Hartmann 6d functions, standard test functions in Bayesian optimization[2] [14]. These functions are continuous, non-convex, multimodal, evaluated in a box, and two, three, and six dimensional, respectively. For each function we consider queries with additive Gaussian noise with standard deviation $\frac{1}{2}$ and 5. For each algorithm we run a series of trials where we fix the budget on samples and run the algorithm. The settings of the algorithm parameters, and number of trials are as follows:

- *Bayesian Optimization*: Given a fixed budget, we choose the number of samples per iteration so that the number of iterations (i.e. number of experiments) matched the number of rounds

[2]Details, and implementation, and a visualization of this function at `www.sfu.ca/ ssurjano/branin.html`, `www.sfu.ca/ ssurjano/branin.html`, `www.sfu.ca/ ssurjano/hart3.html`, and `www.sfu.ca/ ssurjano/hart6.html`, respectively

Hyperband uses at that budget. As a learning exercise, we use our own implementation of Gaussian processes and Bayesian Optimization[3]. Due to time constraints, we use a squared exponential kernel with hyperparameters chosen to maximize the marginal likelihood and expected improvement as the acquisition function. We also use genetic optimization software to maximize the mean function and the acquisition function [15]. These choices are not the choices that state-of-the-art Bayesian optimization packages make, so presumably the performance we see is worse than using one of those packages. We run 10 trials per budget level. Additionally, due to the time constraint and the time Bayesian optimization takes in higher dimensions, we were unable to get a result for the Hartmann 6d function with high noise.

- *Sequential Halving*: For a fixed budget T, we choose the maximal number of treatments possible, i.e. we choose n such that the number of times each treatment is pulled in the first round satisfies $\left\lfloor \frac{T}{n\lceil \log_2 n \rceil} \right\rfloor = 1$. We run 100 trials per budget level.
- *Hyperband*: We choose $\eta = 4$. We run 100 trails per budget level.
- *Sequential Tree*: For a fixed budget T we choose the number of elements in the initial partition to be $\lfloor \frac{T}{20} \rfloor$ and from the empirical result in the previous section choose $\eta = 2$. We run 50 trials per budget level.
- *Partition Tree*: For budget T we choose the number of rounds to be $\lfloor \sqrt{\frac{T}{40}} \rfloor$. We also add an exit condition if the total volume of the remaining sets becomes smaller than the square root of machine precision. In practice most trials exited before reaching the maximum number of rounds, which means that we choose the rounds to scale too quickly with T; finding the right value of the scaling has proven a challenge, and perhaps underlies the results we see. We run 50 trials per budget level.

Figure 4 shows the results of these trials. There are a number of interesting things to notice.

- Across all settings, Sequential Halving with maximal n performs the same or better than Hyperband. Perhaps this is not surprising. Hyperband was designed for the non-stochastic setting of hyperparameter optimization. Since we do not know how the loss scales with the number of resources, Hyperband tries many different values of n. In the noisy Gaussian setting, the basic Chernoff bound tells us that the difference between the empirical mean and the true mean scales like $\frac{1}{\sqrt{t}}$ with high probability. This scaling might imply that the gain in trying more configurations is greater than the loss in sampling each configuration less.
- Bayesian Optimization performs quite poorly. Even for the largest budget in the trial, Hyperband has only 38 rounds. This means that Bayesian optimization could only see 38 different treatments, so it is no surprise that it performed poorly. One could increase the number of iterations that Bayesian optimization uses while decreasing the number of samples per iteration; however, in the experimental setting this corresponds to running many small experiments, which is usually much more costly than running a single large experiment.
- Sequential Tree performs well in the low noise setting. The algorithm particularly outperforms Sequential Halving in the low to moderate budget setting. However; once the noise increases Sequential Tree has an order of magnitude worse performance. If we know or can estimate that the noise is small, then perhaps Sequential Tree would be a good choice.
- Partition Tree is robust to noise. In Section 4 we describe that robustness to over-fitting was the major motivation of Partition Tree, and empirically we see that borne out. In fact, Partition Tree performs as good or better than the other algorithms in almost all settings. However, the algorithm clearly does not take advantage of increased budget, as in most cases empirically the rate is nearly flat. In future work we would like to explore if any changes to the algorithm can fix this problem and lead to a superior algorithm.
- The two bandit-based random search algorithms have better rates of convergence than the two tree-based partitioning algorithms. Noting that the slope a of a log-log plot corresponds

[3]Code for Gaussian processes can be found at `github.com/ebenmichael/gaussianProcess` and code for all optimization algorithms can be found at `github.com/ebenmichael/bandit_experiments`

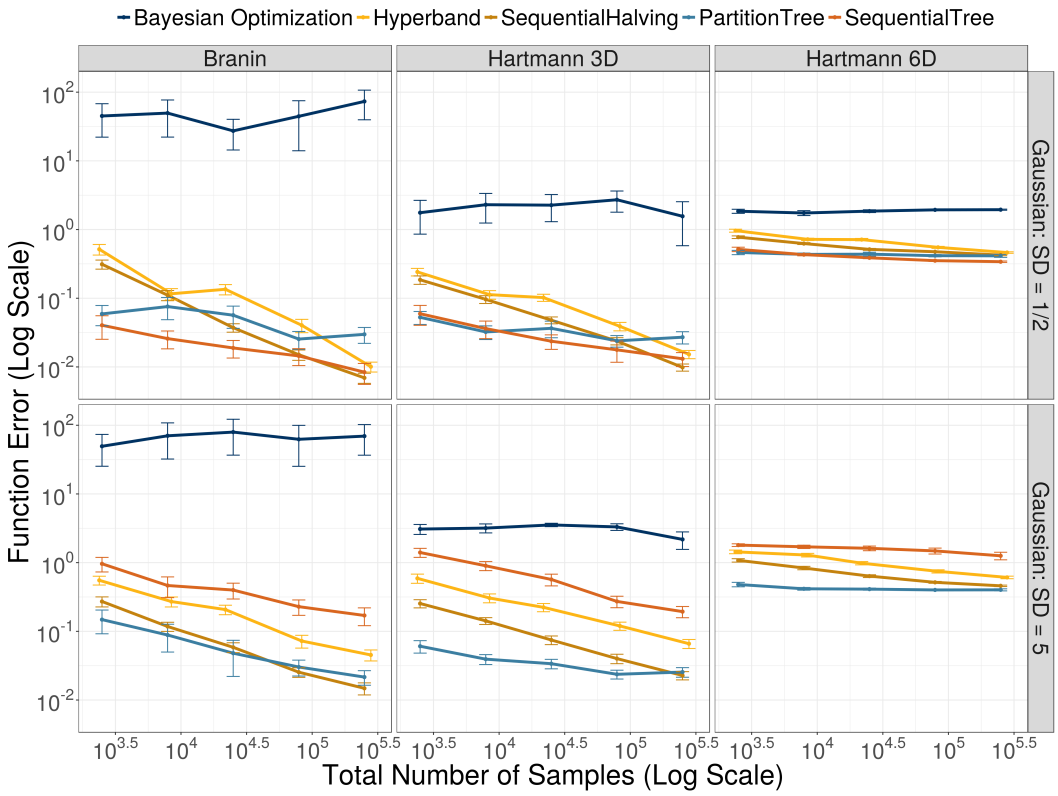


Figure 4: We run each algorithm for each combination of function and noise level and budget several times in order to asses their performance in the test cases

to $y = x^a$, we can visually see the difference in slopes. To get a more precise look at this we regress the log of the objective error on the log of the budget, and place the results in Table 1. Both Figure 4 and Table 1 show the curse of dimensionality: as the dimension grows, the rates flatten out considerably. The table also confirms that the bandit-based algorithms have superior rates empirically.

Algorithm	Branin		Hartmann 3D		Hartmann 6D	
	$\sigma = \frac{1}{2}$	$\sigma = 5$	$\sigma = \frac{1}{2}$	$\sigma = 5$	$\sigma = \frac{1}{2}$	$\sigma = 5$
Hyperband	-0.75	-0.56	-0.57	-0.45	-0.14	-0.18
Sequential Halving	-0.83	-0.65	-0.62	-0.54	-0.12	-0.18
Partition Tree	-0.19	-0.31	-0.12	-0.16	-0.02	-0.03
Sequential Tree	-0.28	-0.38	-0.29	-0.48	-0.08	-0.10

Table 1: We estimate the empirical rate of convergence for each algorithm (not including Bayesian optimization) on each problem by regressing the log error on the log budget

5 Conclusion and Future Directions

The results from the previous section are promising in a few ways. First, Sequential Halving works quite well in higher budget settings and it has the benefit of being simple to understand and implement. Second, although Sequential Tree and Partition Tree have empirically worse rates, and Sequential Tree empirically does poorly in high noise scenarios, at least one of these two algorithms gives performance an order of magnitude better than the other algorithms in all low budget settings. Equivalently, with over an order of magnitude lower budgets, these algorithms have the same performance as Sequential Halving. For future work, we would like to explore further variants of these algorithms, or algorithms with a similar flavor which incorporate configuration selection and evaluation in a different way;

for example, one possible algorithm could be to sample new treatments uniformly over a small neighborhood of the treatments which empirically do well.

For more future work we would also like to consider a different interpretation of the problem of finding the best treatment. We have several times noted that a single large experiment is possibly cheaper than many small experiments; rather than fix the budget on experimental units/samples, we could fix the number of samples per experiment and put a budget on the number of experiments. There is work in best-arm identification in multi-armed bandits under batch pulls that may prove fruitful to explore [16].

References

[1] A. Agarwal, D. P. Foster, D. Hsu, S. M. Kakade, and A. Rakhlin, "Stochastic convex optimization with bandit feedback," *SIAM Journal on Optimization*, vol. 23, no. 1, pp. 213–240, 2013.

[2] O. Shamir, "On the complexity of bandit and derivative-free stochastic convex optimization," *Colt*, vol. 30, pp. 1–22, 2013.

[3] L. Li, K. Jamieson, G. DeSalvo, A. Rostamizadeh, and A. Talwalkar, "Hyperband: A Novel Bandit-Based Approach to Hyperparameter Optimization," *arXiv preprint*, 2016.

[4] E. Brochu, V. M. Cora, and N. De Freitas, "A tutorial on Bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning," *ArXiv*, p. 49, 2010.

[5] J. Snoek, H. Larochelle, and R. Adams, "Practical Bayesian Optimization of Machine Learning Algorithms.," *NIPS*, 2012.

[6] C. E. Rasmussen and C. K. I. Williams, *Gaussian processes for machine learning.*, vol. 14. 2004.

[7] I. Murray and R. P. Adams, "Slice sampling covariance hyperparameters of latent Gaussian models," *Advances in Neural Information Processing ...*, vol. 2, no. 1, p. 9, 2010.

[8] N. Srinivas, A. Krause, S. M. Kakade, and M. Seeger, "Gaussian Process Optimization in the Bandit Setting: No Regret and Experimental Design," *Proceedings of the 27th International Conference on Machine Learning (ICML 2010)*, pp. 1015–1022, 2010.

[9] R. B. Gramacy, H. K. H. Lee, C. Holmes, and M. Osborne, "Optimization Under Unknown Constraints," in *Bayesian Statistics 9*, 2011.

[10] Z. Karnin, T. Koren, and O. Somekh, "Almost optimal exploration in multi-armed bandits," *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, vol. 28, pp. 1238–1246, 2013.

[11] K. Jamieson and A. Talwalkar, "Non-stochastic Best Arm Identification and Hyperparameter Optimization," in *Proceedings of AISTATS*, 2015.

[12] E. Even-dar, S. Mannor, and Y. Mansour, "Action Elimination and Stopping Conditions for Reinforcement Learning," *ICML*, vol. 7, pp. 1079–1105, 2003.

[13] D. R. Jones, "A Taxonomy of Global Optimization Methods Based on Response Surfaces," *Journal of Global Optimization*, vol. 21, pp. 345–383, 2001.

[14] M. Hoffman, E. Brochu, and N. D. Freitas, "Portfolio Allocation for Bayesian Optimization," *Conference on Uncertainty in Artificial Intelligence*, pp. 327–336, 2011.

[15] W. R. J. Mebane and J. S. Sekhon, "Genetic Optimization Using Derivatives: The rgenoud Package for R," *Journal of Statistical Software*, vol. 42, no. 11, 2011.

[16] K.-S. Jun and K. Jamieson, "Top Arm Identification in Multi-Armed Bandits with Batch Arm Pulls," *AISTATS*, vol. 51, pp. 139–148, 2016.