Batched Stochastic Bayesian Optimization via Combinatorial Constraints Design

Kevin K. Yang[1]
FVL57

Yuxin Chen
Caltech

Alycia Lee
Caltech

Yisong Yue
Caltech

Abstract

In many high-throughput experimental design settings, such as those common in biochemical engineering, batched queries are often more cost effective than one-by-one sequential queries. Furthermore, it is often not possible to directly choose items to query. Instead, the experimenter specifies a set of constraints that generates a library of possible items, which are then selected stochastically. Motivated by these considerations, we investigate *Batched Stochastic Bayesian Optimization* (BSBO), a novel Bayesian optimization scheme for choosing the constraints in order to guide exploration towards items with greater utility. We focus on *site-saturation mutagenesis*, a prototypical setting of BSBO in biochemical engineering, and propose a natural objective function for this problem. Importantly, we show that our objective function can be efficiently decomposed as a difference of submodular functions (DS), which allows us to employ DS optimization tools to greedily identify sets of constraints that increase the likelihood of finding items with high utility. Our experimental results show that our algorithm outperforms common heuristics on both synthetic and two real protein datasets.

1 Introduction

Bayesian optimization is a popular technique for optimizing black-box objective functions, with applications in (sequential) experimental design, parameter tuning, recommender systems and more. In the classical setting, Bayesian optimization techniques assume that items can be directly queried at each iteration. However, in many real-world applications such as those in biochemical engineering, direct querying is not possible: instead, a (constrained) library of items is specified, and then batches of items from the library are stochastically queried.

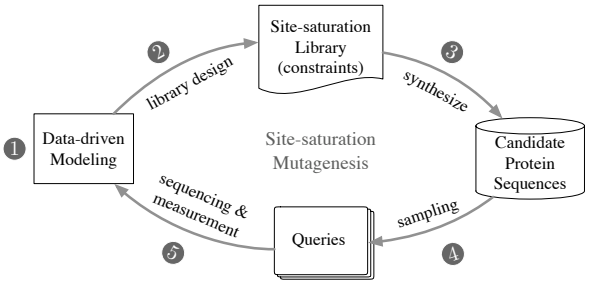


Figure 1: Data-driven site-saturation mutagenesis. (1) Machine learning model for predicting certain protein properties; (2) site-saturation library design; (3) synthesize protein sequences according to the site-saturation libraries; (4) randomly sample proteins for sequencing; (5) sequence and measure the properties of the sampled proteins.

As a prototypical example in biochemical engineering, let us consider *site-saturation mutagenesis* (SSM) (Voigt et al., 2001), a protein-engineering strategy that mutates a small number of critical sites in a protein sequence (cf. Fig. 1). At each round, a combinatorial library is designed by specifying which amino acids are allowed at the specified sites (step (1-3)), and then a batch of amino acid sequences from the library is sampled with replacement (step 4). The sampled sequences are evaluated for their ability to perform a desired function (step 5), such as a chemical reaction.

Ideally, at each iteration, the amino acids to be considered at each site should be chosen to maximize the number of improved sequences expected in the stochastic batch sample from the resulting library. Finding such libraries is highly non-trivial: it requires solving a combinatorial optimization problem over an exponen-

[1]Research done while author was at Caltech.

Proceedings of the 22[nd] International Conference on Artificial Intelligence and Statistics (AISTATS) 2019, Naha, Okinawa, Japan. PMLR: Volume 89.

tial number of items. Libraries are designed by choosing the allowed amino acids at each site ('constraints') from the set of all amino acids at all sites. Adding allowed constraints results in an exponential number of items in the library. As mentioned above, these challenges are exacerbated due to the uncertainty from sampling batches of queries. Thus, new optimization schemes and algorithmic tools are needed for addressing such problems.

Our contribution In this paper, we investigate *Batched Stochastic Bayesian Optimization* (BSBO), a novel Bayesian optimization scheme for choosing a library design in order to guide exploration towards items with greater utility. This scheme is unique in that we choose a library design instead of directly querying items, and the items are queried in stochastic batches (e.g. 10-1000 items per batch). In particular, we focus on library design for site-saturation mutagenesis, and identify a natural objective function that evaluates the quality of a library design given the current information about the system. We propose Online-DSOpt, an efficient online algorithm for optimization over stochastic batches. In a nutshell, Online-DSOpt assembles each batch by decomposing the objective function into the difference of two submodular functions (DS). This allows us to employ DS optimization tools (e.g., Narasimhan & Bilmes (2005)) to greedily identify sets of constraints that increase the likelihood of finding items with high utility. We demonstrate the performance of Online-DSOpt on both synthetic and two experimentally-generated protein datasets, and show that our algorithm in general outperforms conventional greedy heuristics and efficiently finds rare, highly-improved, sequences.

2 Related Work

Bayesian optimization with Gaussian processes Our work addresses a specific setting for Gaussian process (GP) optimization. GPs are infinite collections of random variables such that every finite subset of random variables has a multivariate Gaussian distribution. A key advantage of GPs is that it is very efficient to perform inference, which makes it one of the most popular theoretical tools for Bayesian optimization (Rasmussen & Williams, 2006; Srinivas et al., 2010; Wang et al., 2016). Notably, Srinivas et al. (2010) introduce the Gaussian Process Upper Confidence Bound (GP-UCB) algorithm for Bayesian Bandit optimization, which provides bounds on the cumulative regret when sequentially querying items. Desautels et al. (2014) generalize this to batch queries. In contrast to our setting, these algorithms require the ability to *directly* query items, either sequentially or in batches.

GP optimization for protein engineering GP-UCB has been used to find improved protein sequences when sequences can be queried directly (Romero et al., 2013; Bedbrook et al., 2017). GPs (Saito et al., 2018) and other machine-learning methods (Wu et al., 2019) have been used to select constraints for SSM libraries. However, previous work relied on ad-hoc heuristics and do not provide a general procedure for selecting constraints in a model-driven way.

Information-parallel learning In addition to the bandit setting (Desautels et al., 2014), there is a large body of literature on various machine learning settings that exploit information-parallelism. For example, in large-scale optimization, mini-batch/parallel training has been extensively explored to reduce the training time of stochastic gradient descent (Li et al., 2014; Zinkevich et al., 2010). In batch-mode active learning (Hoi et al., 2006; Guillory & Bilmes, 2010; Chen & Krause, 2013), an active learner selects a set of examples to be labeled simultaneously. The motivation behind batch active learning is that in some cases it is more cost-effective to request labels in large batches, rather than one-at-a-time. This setting is also referred to as buy-in-bulk learning (Yang & Carbonell, 2013). In addition to the simpler modeling assumption of being able to directly issue queries, these approaches also differ from our setting in terms of the objective: the batch-mode active learning algorithms aim to find a set of items that are maximally informative about some target hypothesis (hence to maximally explore), whereas we want to identify the best item (i.e., to both explore and exploit).

Submodularity and DS optimization Submodularity (Nemhauser et al., 1978) is a key tool for solving many discrete optimization problems, and has been widely recognized in recent years in theoretical computer science and machine learning. While a growing number of previously studied problems can be expressed as submodular minimization (Jegelka & Bilmes, 2011) or maximization (Kempe et al., 2003; Krause & Guestrin, 2007) problems, standard maximization and minimization formulations only capture a small subset of discrete optimization problems. Narasimhan & Bilmes (2005) show that any set function q can be decomposed as the difference of two submodular functions h and g. Replacing h with its modular upper bound, g with its modular lower bound, or both reduces the problem of minimizing q to a series of submodular minimizations, submodular maximizations, or modular minimizations, respectively, that are guaranteed to reduce q at every iteration and to arrive at a local minimum of q (Iyer & Bilmes, 2012). In general, computing a DS decomposition requires exponential time. We present two polynomial-time de-

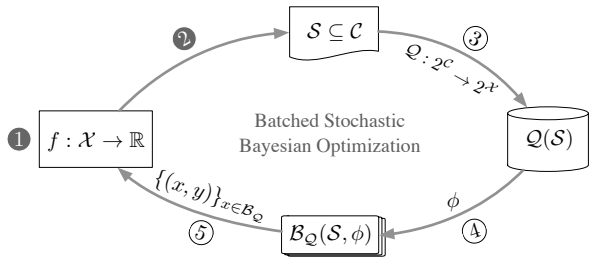


Figure 2: The batched stochastic Bayesian optimization setting. (1) Bayesian modeling (2) combinatorial constraints design; (3) candidate query generation; (4) random sampling; (5) batched queries.

compositions of our objective function.

3 Problem Statement

3.1 Problem Setup

We aim to optimize a black box utility function, $f : \mathcal{X} \to \mathbb{R}$. In contrast to classical Bayesian optimization which sequentially queries the function value $f(x_t)$ for an selected item $x_t \in \mathcal{X}$, we assume that the experimenter can only choose a subset of constraints (i.e., rules for generating items) from a ground set $\mathcal{C}$, based on which a stochastic batch of items are generated and measured. More concretely, we consider the following interactive protocol, as illustrated in Fig. 2. At each round the following happens:

- The algorithm chooses a set of constraints $\mathcal{S} \subseteq \mathcal{C}$ based on current knowledge of f (Fig. 2, step (2)).
- The chosen constraints are used to construct a *library* of candidate queries: $Q(\mathcal{S}) \subseteq \mathcal{X}$, where $Q : 2^{\mathcal{C}} \to 2^{\mathcal{X}}$ denotes the physical process that produces items under these constraints (Fig. 2, step (3)).
- A batch of n queries $\mathcal{B}_Q(\mathcal{S}, \phi) \subseteq \mathcal{X}$ is randomly selected from the library $Q(\mathcal{S})$ via a stochastic sampling procedure (Fig. 2, step (4)). Here, ϕ represents the random state of the sampling procedure $\mathcal{B}_Q$.
- When querying each item $x \in \mathcal{B}_Q(\mathcal{S}, \phi)$, we get to observe the function value there, perturbed by noise: $y = f(x) + \varepsilon(x)$. Here, the noise $\varepsilon(x) \sim \mathcal{N}(0, \sigma^2)$ represent i.i.d. Gaussian white noise. We further assume that querying each item x achieves reward $f(x)$ and incurs some cost $c(\{x\})$, where $c : 2^{\mathcal{X}} \to \mathbb{R}$ denotes the cost function of a set of items (Fig. 2, step (5)).
- The results of the queries are used to update f.

We model f as a sample from a Gaussian process, denoted by $f \sim \mathrm{GP}(\mu(x), k(x, x'))$. Suppose that we have queried $\mathcal{A} \subseteq \mathcal{X}$ and received $\mathbf{y}_{\mathcal{A}} = [f(x_i) + \varepsilon(x_i)]_{x_i \in \mathcal{A}}$ observations. We can obtain the posterior mean $\mu_{\mathcal{A}}(x)$ and covariance $k_{\mathcal{A}}(x, x')$ of the function through the covariance matrix $K_{\mathcal{A}} = [k(x_i, x_j) + \sigma^2 \mathbb{1}]_{x_i, x_j \in \mathcal{A}}$ and $\mathbf{k}_{\mathcal{A}}(x) = [k(x_i, x)]_{x_i \in \mathcal{A}}$:

$$\mu_{\mathcal{A}}(x) = \mu(x) + \mathbf{k}_{\mathcal{A}}(x)^\intercal K_{\mathcal{A}}^{-1} \mathbf{y}_{\mathcal{A}},$$
$$k_{\mathcal{A}}(x, x') = k(x, x') - \mathbf{k}_{\mathcal{A}}(x)^\intercal K_{\mathcal{A}}^{-1} \mathbf{k}_{\mathcal{A}}(x').$$

3.2 The Objective

Simple regret Our overall goal is to minimize the (expected) simple regret, defined as $R_t(x) = f(x^*) - f(x_t)$ over T rounds, where $x^* = \arg\max_{x \in \mathcal{X}} f(x)$ is the item of the maximum utility. In other words, we aim to maximize the reward in order to converge to performing as well as x^* as efficiently as possible. We refer to this problem as the batched stochastic Bayesian optimization (BSBO) problem.[2]

Acquisition function Assume that we have a budget of querying n items for each batch of experiments and that each batch $\mathcal{B}_Q$ is selected by sampling uniformly from the library. At each iteration, we wish to select the constraints $\mathcal{S}$ that will maximize the (expected) number of improved items observed in the next stochastic batched query. If the current best item has a value τ, then we seek a set of constraints $\mathcal{S}^* \in \arg\max F(\mathcal{S})$, where:

$$F(\mathcal{S}) = \mathbb{E}_\phi \left[\sum_{x \in \mathcal{B}_Q(\mathcal{S})} \mathbb{1}(f(x) > \tau) \right]. \tag{3.1}$$

Here, $\mathbb{1}$ is the indicator function. This objective is intractable under the GP posterior, as the dependencies between $f(x)$ preclude a closed form. We ignore the dependencies between the utilities to arrive at the following surrogate function;

$$\hat{F}(\mathcal{S}) = \sum_{x \in Q(\mathcal{S})} \rho(x) \left[1 - \left(1 - \frac{1}{|Q(\mathcal{S})|} \right)^n \right]. \tag{3.2}$$

The rewards $\rho(x) = P(f(x) > \tau)$ can be computed for all $x \in Q(\mathcal{C})$ from the GP posterior for each item using the Gaussian survival function by ignoring off-diagonal entries in the predictive posterior covariance. Note that this surrogate acquisition function $\hat{F}$ captures the expected reward under an *independence* assumption. As we will demonstrate later in §5.3, despite such an assumption, we observe a strong correlation between $\hat{F}$ and F on the experimental datasets

[2] When the set of candidate queries $Q(\mathcal{S}_t)$ contains only *one* unique item at each round t, then the BSBO problem reduces to the standard Bayesian optimization problem.

Algorithm 1: Online Batched Constraints Design via DS Optimization (Online-DSOpt)

Input: Constraints set $\mathcal{C} = \bigcup_{\ell=1}^{L} \mathcal{C}^{(\ell)}$; number of rounds T; budget on each batch n; GP prior on f
begin
  $\mathcal{A} \leftarrow \emptyset$
  /* iteratively select the next batch */
  for t *in* $1, \ldots, T$ **do**
    /* compute the reward matrix $M = \{f(x)\}$ */
    $M \leftarrow$ ComputeReward(posterior on $f, \mathcal{A}$)
    $\mathcal{S} \leftarrow$ DSOpt$(\mathcal{C}, M, n)$
    /* posterior update */
    $\mathcal{A} \leftarrow \mathcal{A} \cup \mathcal{B}_Q(\mathcal{S}, \phi)$
  Output: Optimizer of f

we study, which are known to have high dependencies between the rewards $\rho(x)$ of different items.

3.3 Site-Saturation Library Design

We now consider the site-saturation library design problem as a special case of BSBO. In site-saturation mutagenesis, the utility function $f(x)$ specifies the utility of a protein sequence x, and the constraint set $\mathcal{C} = \bigcup_{\ell=1}^{L} \mathcal{C}^{(\ell)}$ specifies the set of amino acids allowed at each site of the protein sequence. L denotes the number of sites, and $\mathcal{C}^{(\ell)}$ denotes the set of all possible amino acids[4] at site ℓ. We denote the set of amino acids selected for site ℓ by $\mathcal{S}^{(\ell)}$; hence $\mathcal{S} = \bigcup_{\ell=1}^{L} \mathcal{S}^{(\ell)}$. The candidate query pool (library) Q consists of all possible protein sequences that can be generated w.r.t. the constraints: $\mathcal{S}$:

$$Q(\mathcal{S}) = \prod_{\ell=1}^{L} \mathcal{S}^{(\ell)}. \tag{3.3}$$

Note that adding constraints generally *increases* the number of allowed items.

4 Algorithms

We now present Online-DSOpt (Algorithm 1), an online learning framework for (online) batched stochastic Bayesian optimization. Our framework relies on a novel discrete optimization subroutine, DSOpt, which aims to maximize the expected reward for each batched experiment. At each iteration, Online-DSOpt uses a GP trained on previously-observed items to compute the reward for each item $x \in Q(\mathcal{C})$ (cf., Line 3) and then invokes DSOpt to select constraints. Pseudocode for DSOpt is presented in Algorithm 2. A batch of items is then sampled stochastically from the resulting library and used to update the GP.

[4] $\mathcal{C}^{(\ell)}$ is typically the 20 canonical amino acids.

Algorithm 2: DS Optimization (DSOpt)

Input: Constraints set $\mathcal{C} = \bigcup_{\ell=1}^{L} \mathcal{C}^{(\ell)}$; reward matrix $M = \{f(x)\}$; budget on each batch n
begin
  Set up $\hat{F}$ from the inputs (M, n)
  /* Decompose $\hat{F}$ into diff of submod funcs. */
  $h, g \leftarrow$ DSConstruct-SA$(\hat{F}, \mathcal{C})$
  (or $h, g \leftarrow$ DSConstruct-DC$(\hat{F}, \mathcal{C})$)
  /* Initiliaze the starting position */
  $\mathcal{S}_{\text{cand}} \leftarrow \emptyset$
  $\mathcal{S} \leftarrow$ Init$(\mathcal{C})$
  /* Optimize $h - g$ using ModMod or SupSub. */
  while $\mathcal{S}$ *not converged* **do**
    /* Keep track of local search solutions */
    $\mathcal{S}_{\text{cand}} \leftarrow \mathcal{S}_{\text{cand}} \cup$ LocalSearch$(\hat{F}, \mathcal{S}, \mathcal{C})$
    /* Make a greedy move from $\mathcal{S}$ */
    $\mathcal{S} \leftarrow$ ModMod$(h - g, \mathcal{S}, \mathcal{C})$
    (or $\mathcal{S} \leftarrow$ SupSub$(h - g, \mathcal{S}, \mathcal{C})$)
  $\mathcal{S}_{\text{cand}} \leftarrow \mathcal{S}_{\text{cand}} \cup \{\mathcal{S}\}$
  /* pick the best among candidate solutions */
  $\mathcal{S}^* \leftarrow \arg\min_{\mathcal{S} \in \mathcal{S}_{\text{cand}}} \left\{\hat{F}(\mathcal{S})\right\}$
  Output: Set of selected constraints $\mathcal{S}^*$

A key component of the DS optimization subroutine DSOpt is a DS decomposition of the objective. Note that in general, finding a DS decomposition of an arbitrary set function requires searching through a combinatorial space and can be computationally prohibitive. As one of our main contributions, we present two polynomial time algorithms, DSConstruct-SA (Algorithm 3) and DSConstruct-DC (Algorithm 4), for decomposing our surrogate objective. Both algorithms exploit the structure of the objective function: DSConstruct-SA decomposes the objective via submodular augmentation (Narasimhan & Bilmes, 2005); DSConstruct-DC decomposes the objective via a difference of convex functions (DC) decomposition.

4.1 DS Optimization

After obtaining the submodular decomposition $\hat{F} = -(h - g)$, DSOpt (Algorithm 2) proceeds to greedily optimize the DS function. For example, let us consider running the Modular-modular procedure (ModMod) (Iyer & Bilmes, 2012) for making a greedy move at Line 7 of Algorithm 2. Since our goal is to maximize $-(h - g)$ (i.e., to minimize $h - g$), we will seek to minimize the upper bound on $h - g$. The ModMod procedure constructs a modular upper bound on the first submodular component, denoted by $\mathsf{ub}^h \geq h$, and a

Algorithm 3: DS Construction via Submodular Augmentation (DSConstruct-SA)

Input: Constraints set $\mathcal{C} = \bigcup_{\ell=1}^{L} \mathcal{C}^{(\ell)}$; surrogate objective function $\hat{F}$, budget on each batch n; selected constraints $\mathcal{S}$
begin
  $v(x) \leftarrow \sqrt{x}$ for $x \in \{1, \dots, |\mathcal{C}|\}$
  $\alpha \leftarrow v(n-2) + v(n) - 2v(n-1)$
  /* compute β' of Eq.(4.2) */
  foreach $x \in \{1, \dots, |Q(\mathcal{C})|\}$ **do**
    $r_1(x) \leftarrow \left(1 - \frac{1}{s}\right)^n - \left(1 - \frac{1}{2s}\right)^n$
    $r_2(x) \leftarrow \max_{\mathcal{T}:|\mathcal{T}|\le s} \sum_{x\in\mathcal{T}} f(x)$
  $\beta' \leftarrow -\max_x r_1(x) r_2(x)$
  $h_1(\mathcal{S}) \leftarrow \frac{|\beta'|}{\alpha} v(|\mathcal{S}|)$
  $g_1(\mathcal{S}) \leftarrow \hat{F}(\mathcal{S}) + \frac{|\beta'|}{\alpha} v(|\mathcal{S}|)$
  Output: DS decomposition $\hat{F} = -(h_1 - g_1)$

modular lower bound on the second submodular component, denoted by $\mathsf{lb}^g \le g$. Both modular bounds are tight at the current solution $\mathcal{S}$: $\mathsf{ub}^h(\mathcal{S}) = h(\mathcal{S})$, $\mathsf{lb}^g(\mathcal{S}) = g(\mathcal{S})$. ModMod then tries to solve the following optimization problem, starting from $\mathcal{S}$:

$$\mathcal{S}^* \in \arg\min_{\mathcal{S}} \left(\mathsf{ub}^h(\mathcal{S}) - \mathsf{lb}^g(\mathcal{S})\right).$$

To ensure that we find a better solution, we augment the ModMod procedure with a sequence of additional local search solutions, and in the end pick the best among all. The local search procedure, LocalSearch (cf. Line 6 of Algorithm 2), sequentially makes greedy steps (by adding or removing a constraint from the current solution) until no further action is improving the current solution. The following theorem states that our DS optimization subroutine DSOpt is guaranteed to find a "good" solution:

Theorem 1 (Adapted from Iyer & Bilmes (2012))**.** *Algorithm 2 is guaranteed to find a set of constraints that achieves a local maximum of $\hat{F}$.*

4.2 DS Construction via Submodular Augmentation

It is well-established that every set function can be expressed as the sum of a submodular and a supermodular function (Narasimhan & Bilmes, 2005). In particular, Iyer & Bilmes (2012) provide the following constructive procedure for decomposing a set function into the DS form: Given a set function q, one can define $\beta = \min_{\mathcal{S}\subseteq\mathcal{S}'\subseteq\mathcal{C}\setminus j} \Delta_q(j \mid \mathcal{S}) - \Delta_q(j \mid \mathcal{S}')$, where $\Delta_q(j \mid \mathcal{S}) := q(\mathcal{S}\cup\{j\}) - q(\mathcal{S})$ denotes the gain of adding j to $\mathcal{S}$. When q is not submodular, we know that $\beta < 0$. Now consider any strictly submodular function p, with $\alpha = \min_{\mathcal{S}\subseteq\mathcal{S}'\subseteq\mathcal{C}\setminus j} \Delta_p(j \mid \mathcal{S}) - \Delta_p(j \mid \mathcal{S}') > 0$. Define $h(\mathcal{S}) = q(\mathcal{S}) + \frac{|\beta'|}{\alpha} p(\mathcal{S})$ for any $\beta' < \beta$. It is easy to verify that h is submodular since $\min_{\mathcal{S}\subseteq\mathcal{S}'\subseteq\mathcal{C}\setminus j} \Delta_h(j \mid \mathcal{S}) - \Delta_h(j \mid \mathcal{S}') \ge \beta + |\beta'| \ge 0$. Hence $q(\mathcal{S}) = h(\mathcal{S}) - \frac{|\beta|}{\alpha} p(\mathcal{S})$ is a difference between two submodular functions.

Algorithm 4: DS Construction via DC Decomposition (DSConstruct-DC)

Input: Constraints set $\prod_{\ell=1}^{L} \mathcal{C}_\ell$; budget on each batch n; selected constraints $\mathcal{S}$
begin
  $u(x) \leftarrow \frac{x^2}{2}$, $\alpha \leftarrow 1$
  $r(x) \leftarrow \left(1 - \frac{1}{x}\right)^n$,
  $\beta \leftarrow |\min_x r''(x)|$ for $x \in \{1, \dots, |Q(\mathcal{C})|\}$.
  $h_2(\mathcal{S}) \leftarrow -\left(1 + \frac{\beta}{\alpha} u(Q(|\mathcal{S}|))\right) \sum_{x\in Q(\mathcal{S})} f(x)$
  $g_2(\mathcal{S}) \leftarrow$
    $-\left(r(|Q(\mathcal{S})|) + \frac{\beta}{\alpha} u(Q(|\mathcal{S}|))\right) \sum_{x\in Q(\mathcal{S})} f(x)$
  Output: DS decomposition $\hat{F} = -(h_2 - g_2)$

We refer to the above decomposition strategy as DSConstruct-SA (where SA stands for "submodular augmentation"), and present the pseudo code in Algorithm 3. As suggested in (Iyer & Bilmes, 2012), we choose the submodular augmentation function $p(\mathcal{S}) = v(|\mathcal{S}|)$, where $v(x)$ is a concave function, and therefore $\alpha = \min_{x\le x';x,x'\subseteq\mathbb{Z}} v(x+1) - v(x) - v(x'+1) - v(x')$. This leads to the following decomposition of our surrogate objective $\hat{F}$:

$$\hat{F}(\mathcal{S}) = \underbrace{\left(\hat{F}(\mathcal{S}) + \frac{|\beta'|}{\alpha} v(|\mathcal{S}|)\right)}_{g_1(\mathcal{S})} - \underbrace{\frac{|\beta'|}{\alpha} v(|\mathcal{S}|)}_{h_1(\mathcal{S})}, \tag{4.1}$$

where h_1, g_1 by construction are submodular functions, and β' is a lower bound on β:

$$\beta' \le \beta = \min_{\mathcal{S}\subseteq\mathcal{S}'\subseteq\mathcal{C}\setminus j} \Delta_{\hat{F}}(j \mid \mathcal{S}) - \Delta_{\hat{F}}(j \mid \mathcal{S}'). \tag{4.2}$$

The key step of the DSConstruct-SA algorithm is to construct such a lower bound β'. The following lemma, which is proved in the Appendix, shows that one can compute β' in polynomial time, and hence can efficiently express $\hat{F}$ as a DS as defined in Eq. (4.1).

Lemma 2. *Algorithm 3 returns a DS-decomposition of $\hat{F}$ in polynomial time.*

4.3 DS Construction via DC Decomposition

We now consider an alternative strategy for decomposing the surrogate function $\hat{F}$, based on a novel construction procedure that reduces to expressing a

continuous function as the difference of convex (DC) functions. Concretely, we note that $\hat{F}(\mathcal{S})$ consists of two (multiplicative) terms: (i) a supermodular set function $\sum_{x\in Q(\mathcal{S})} f(x)$, and (ii) a set function that only depends on the cardinality of the input, i.e., $\sum_{x\in Q(\mathcal{S})} f(x)\left(1-\left(1-\frac{1}{|Q(\mathcal{S})|}\right)^n\right)$. As is further discussed in the Appendix, we show that one can exploit this structure, and focus on the DC decomposition of term (ii). We provide the detailed algorithm in Algorithm 4, and refer to it as DSConstruct-DC (where DC stands for "difference of convex decomposition").

It is easy to check from Algorithm 3 that DSConstruct-SA runs in quadratic time w.r.t. $|Q(\mathcal{C})|$. In contrast, DSConstruct-DC only requires finding the minimum of an array of size $|Q(\mathcal{C})|$, which, in the best case, runs in linear time w.r.t. $|Q(\mathcal{C})|$. At the end of the algorithm, DSConstruct-DC outputs the following DS function:

$$\hat{F}(\mathcal{S}) = \underbrace{\sum_{x\in Q(\mathcal{S})} (-\rho(x)) \cdot \left(r(|Q(\mathcal{S})|) + \frac{\beta}{\alpha} u(Q(|\mathcal{S}|))\right)}_{g_2(\mathcal{S})} - \underbrace{\sum_{x\in Q(\mathcal{S})} (-\rho(x)) \left(1 + \frac{\beta}{\alpha} u(Q(|\mathcal{S}|))\right)}_{h_2(\mathcal{S})}. \quad (4.3)$$

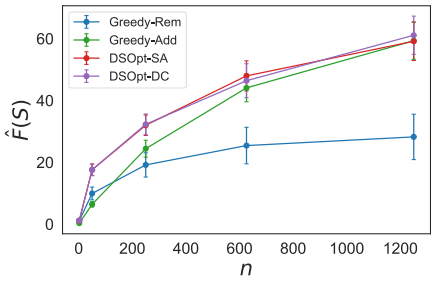
where $u(x)$ is a non-negative, monotone convex function[6], $\alpha = \min_x u''(x)$, $r(x) = \left(1-\frac{1}{x}\right)^n$, and $\beta = |\min_x r''(x)|$. We then prove the following results:

Lemma 3. *With the decomposition as defined in Eq. (4.3), both functions h, g are submodular and hence we obtain a DS-decomposition of $\hat{F}$.*

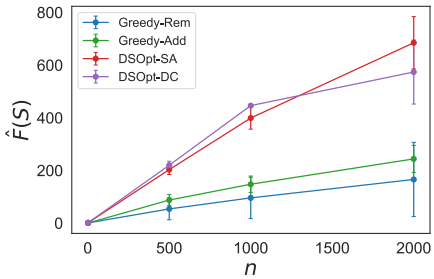
5 Experiments

We empirically evaluate our algorithm on both synthetic and real protein datasets. First, we compare DSOpt against two intuitive greedy heuristics on synthetic datasets (§5.1) and compare online optimization using Online-DSOpt against the two heuristics on a synthetic sequence landscape (§5.2). We then justify the choice of our surrogate objective (Eq. (3.2)) with numerical simulations on real protein datasets in §5.3.2, and demonstrate the performance of Online-DSOpt in §5.3.3. For simplicity, we refer to the two versions of DSOpt as DSOpt-SA (i.e., DSOpt with subroutine DSConstruct-SA) and DSOpt-DC (i.e., DSOpt with subroutine DSConstruct-DC), respectively. We compare against two intuitive greedy search heuristics: Greedy-Add, which greedily adds constraints and Greedy-Rem, which greedily removes constraints until the objective stops improving. Because the empty set is a local optimum (adding any single constraint still results in no valid queries), we must begin the optimization at a set of constraints that yields a non-empty set of queries.

[6]In practice we often set $u(x) = \frac{x^2}{2}$ and thus $\alpha = 1$.



(a) $L = 2$, $|\mathcal{C}^{(\ell)}| = 26\ \forall \ell$



(b) $L = 15$, $|\mathcal{C}^{(\ell)}| = 2\ \forall \ell$

Figure 3: Performance of each algorithm on finding constraints on synthetic datasets. Error bars are standard errors. $\hat{F}$ is the approximate objective, and n is the batch size.

5.1 Synthetic Examples for Batched Optimization

5.1.1 Synthetic Datasets

We evaluate the algorithms on their ability to optimize Eq. (3.2) for two synthetic datasets designed to have multiple local minima. As shown in the appendix, our first synthetic dataset consists of two sites with $|\mathcal{C}^{(1)}| = |\mathcal{C}^{(2)}| = 26$. Values for the items in the library are constructed such that there are disjoint blocks of items with non-zero $\rho(x)$ separated by regions where $\rho(x) = 0$. This guarantees that there are multiple local optima in the constraint space. Similarly, we create a second synthetic dataset with $L = 15$ and $|\mathcal{C}^{(\ell)}| = 2$. The library is structured such that only cells representing subsequences containing specific substrings have non-zero values. Therefore, it is likely that the dataset contains many local optima, which makes it challenging for finding the optimal set of constraints.

5.1.2 Results on Synthetic Datasets

We compare the algorithms across a range of batch sizes n. At each batch size, we initialize each algorithm at $\mathcal{C}$, the constraints that result in the single best query, and 18 randomly selected sets of constraints. Fig. 3a shows the results for the compared algorithms on the first dataset ($L = 2, |\mathcal{C}^{(\ell)}| = 26$), when we vary $n \in [0, 1200]$, and Fig. 3b shows the results on the second dataset ($L = 15, |\mathcal{C}^{(\ell)}| = 2$) for $n \in [0, 2000]$. We observe that DSOpt-SA and DSOpt-DC consistently outperform Greedy-Add and Greedy-Rem across all values of n, under both dataset configurations. At small n, the optima tend to have few constraints, so Greedy-Add performs particularly poorly. As n approaches infinity, the optimum approaches the ground set $\mathcal{C}$, and so Greedy-Rem performs particularly poorly. DSOpt-SA and DSOpt-DC perform very similarly across all values of n. In theory, DSOpt-SA and DSOpt-DC can escape local optima to find better solutions than the local search algorithm (although this appears to be rare on our synthetic datasets).

5.2 Synthetic Experiments for Batched Online Optimization

The objective in the online experiments is to maximize the best item found given a certain budget of evaluations. This is equivalent to minimizing the simple regret described in §3.2 and mimics the objective in biological engineering, where the goal is to find a highly-fit sequence.

5.2.1 Synthetic Sequence Landscape

We evaluate Online-DSOpt on a synthetic dataset generated from an NK model (Kauffman & Weinberger, 1989). NK models generate synthetic sequence landscapes with tuneable nonlinearity. Each sequence x in the landscape has a utility defined as $f(x) = \sum_l^L \tilde{f}_\ell(x_\ell)$ where $\tilde{f}_\ell = f_\ell(x_\ell, x_{k_{\ell 1}}, ..., x_{k_{\ell K}})$ depends on the value of x at position ℓ and K other positions. The utility contributions in this case are chosen randomly from a standard normal distribution, and then we normalize the utility values to have standard deviation 1. We use a landscape with $L = 8$, $|\mathcal{C}^{(\ell)}| = 4$, and $K = 4$.

5.2.2 Results on a Synthetic Sequence Landscape

We test Online-DSOpt on the NK landscape to demonstrate its ability to select constraints that result in libraries enriched in improved sequences. We test Online-DSOpt against a baseline that runs both Greedy-Add and Greedy-Rem at every iteration and picks the best solution found by the two baseline algorithms. For each batch size n, we initiated the experiments with a starting "wild-type" sequence and all its nearest neighbors. We then ran $k = \frac{400}{n} - 1$ iterations of the algorithm with batch size n, resulting in k more batched queries. This simulates an SSM experiment with k rounds of diversification, screening, and selection. At each iteration, we train a GP regression model using a Matérn kernel with $\nu = \frac{5}{2}$ in order to compute the rewards $\rho(x)$. Results for each experiment are shown in Fig. 4. Both Online-DSOpt and the baseline find highly-fit sequences. However, the baseline is more likely to converge to poor local minima, as shown in the case where $n = 100$.

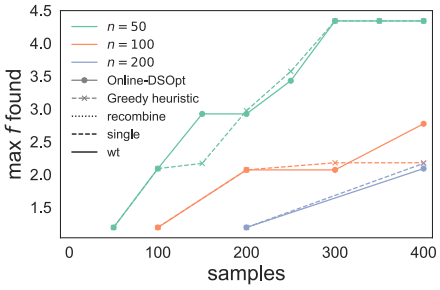
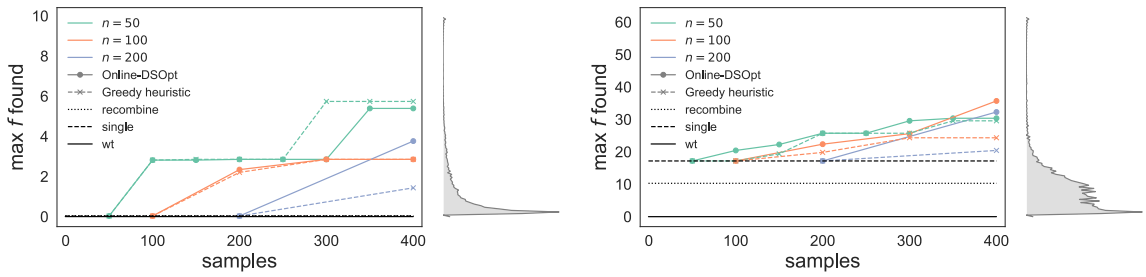


Figure 4: Results on a synthetic landscape generated from an NK model with $L = 8$, $|\mathcal{C}^{(\ell)}| = 4$, and $K = 4$ when running Online-DSOpt or a combination of Greedy-Add and Greedy-Rem. Each colored line corresponds to a different batch size, and every marker corresponds to the maximal value of f found at the end of that batch. The solid horizontal lines show the utilities for the "wild-type" sequences (wt); the dashed horizontal lines show the utilities for the best sequences with exactly one mutation from the wild type (single); and the dotted horizontal lines show the utilities for the sequences that combine the best amino acid at each site determined in the wild-type background (recombine).

5.3 Real Experiments for Batched Online Optimization

5.3.1 Experimental Protein Landscapes

We further evaluate our algorithms on two experimental protein-engineering datasets consisting of measured utility values for every sequence in four-site SSM libraries for protein G domain B1 (GB1) (Wu et al., 2016), an immunoglobulin binding protein, and the protein kinase PhoQ (Podgornaia & Laub, 2015). These fitness landscapes are known to be highly non-additive (mutations have different effects in combination than individually). Having measurements for every fitness value in each library allows us to simulate engineering via multiple rounds of SSM.



(a) GB1 (b) PhoQ

Figure 5: Results on (a) GB1 and (b) PhoQ when running Online-DSOpt or a combination of Greedy-Add and Greedy-Rem. Each colored line corresponds to a different batch size, and every marker corresponds to the maximal value of f found at the end of that batch. The solid horizontal lines show the utilities for the wild-type sequences (wt); the dashed horizontal lines show the utilities for the best sequences with exactly one mutation from the wild type (single); and the dotted horizontal lines show the utilities for the sequences that combine the best amino acid at each site determined in the wild-type background (recombine). The plots on the right show the probability mass of each utility value in the corresponding dataset.

5.3.2 Suitability of the Surrogate Objective

Online-DSOpt uses a GP posterior to model the unobserved utilities. However, there is no closed form for the true batch constraint design objective F as defined in Eq. (3.1), which is to choose constraints $\mathcal{S}$ that maximize the expected number of improved observations found by querying $Q(\mathcal{S})$. The surrogate objective function $\hat{F}$ (Eq. (3.2)) overestimates the true objective because it ignores dependencies between items. To test the suitability of the surrogate objective function, we selected an initial batch of sequences from each of the protein datasets consisting of all the single mutants plus 100 randomly-selected sequences, trained a GP regression model, and used the posterior to compute the rewards $\rho(x)$. In the appendix of this paper, we plot the values of the surrogate objective function $\hat{F}$ against the Monte Carlo estimates of the true objective values F for the two protein datasets. We observe that the independence assumption leads to overestimating the number of improved sequences that will be found, but the values for the surrogate are well-correlated with the true objective.

5.3.3 Results on Real Protein Landscapes

We test Online-DSOpt on the PhoQ and GB1 datasets to demonstrate its ability to select constraints that result in libraries enriched in improved sequences using the procedure described in §5.2. For both GB1 and PhoQ, the probability density function plots in Fig. 5a and Fig. 5b show that Online-DSOpt finds rare sequences that would be extremely difficult to find by randomly sampling < 500 sequences from the entire library. Importantly, it finds much better sequences than combining the best mutation at each site in the wild-type background or the best sequence with a single mutation from the wild-type, as shown in Fig. 5a and Fig. 5b. These are common experimental heuristics for dealing with multi-site SSM libraries where the library is too large to reasonably screen. The combined Greedy-Add and Greedy-Rem baseline performs comparably on these datasets for most values of n, but occasionally converges to poorer optima.

6 Conclusion

In this paper, we investigated a novel Bayesian optimization problem: batched stochastic Bayesian optimization. This problem setting poses two unique challenges: optimizing over the space of constraints instead of directly over items and stochastic sampling. We proposed an effective online optimization framework for searching through the combinatorial design space of constraints in order to maximize the expected number of improved items sampled at each iteration. In particular, we proposed a novel approximate objective function that links a model trained on the individual items to the constraint space and derived two efficient DS decompositions for this objective. Our method efficiently finds sequences with improved fitnesses in fully-characterized SSM libraries for the proteins GB1 and PhoQ, demonstrating its potential to enable engineering via simultaneous SSM even in cases where it is not feasible to measure more than a tiny fraction of the sequences in the library.

Acknowledgments

This work was supported in part by the Donna and Benjamin M. Rosen Bioengineering Center, the

U.S. Army Research Office Institute for Collaborative Biotechnologies, NSF Award #1645832, Northrop Grumman, Bloomberg, PIMCO, and a Swiss NSF Early Mobility Postdoctoral Fellowship.

References

Bedbrook, C. N., Yang, K. K., Rice, A. J., Gradinaru, V., and Arnold, F. H. Machine learning to design integral membrane channelrhodopsins for efficient eukaryotic expression and plasma membrane localization. *PLoS computational biology*, 13(10):e1005786, 2017. 2

Chen, Y. and Krause, A. Near-optimal batch mode active learning and adaptive submodular optimization. In *International Conference on Machine Learning (ICML)*, June 2013. 2

Desautels, T., Krause, A., and Burdick, J. W. Parallelizing exploration-exploitation tradeoffs in gaussian process bandit optimization. *The Journal of Machine Learning Research*, 15(1):3873–3923, 2014. 2

Guillory, A. and Bilmes, J. A. Interactive submodular set cover. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pp. 415–422, 2010. 2

Hoi, S. C. H., Jin, R., Zhu, J., and Lyu, M. R. Batch mode active learning and its application to medical image classification. In *Proceedings of the 23rd international conference on Machine learning*, ICML '06, pp. 417–424, New York, NY, USA, 2006. ACM. ISBN 1-59593-383-2. 2

Iyer, R. and Bilmes, J. Algorithms for approximate minimization of the difference between submodular functions, with applications. In *Proceedings of the Twenty-Eighth Conference on Uncertainty in Artificial Intelligence*, pp. 407–417. AUAI Press, 2012. 2, 4, 5

Jegelka, S. and Bilmes, J. Submodularity beyond submodular energies: coupling edges in graph cuts. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2011)*, pp. 1897–1904. IEEE, 2011. 2

Kauffman, S. A. and Weinberger, E. D. The nk model of rugged fitness landscapes and its application to maturation of the immune response. *Journal of theoretical biology*, 141(2):211–245, 1989. 7

Kempe, D., Kleinberg, J., and Tardos, É. Maximizing the spread of influence through a social network. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 137–146. ACM, 2003. 2

Krause, A. and Guestrin, C. Near-optimal observation selection using submodular functions. In *AAAI*, volume 7, pp. 1650–1654, 2007. 2

Li, M., Zhang, T., Chen, Y., and Smola, A. J. Efficient mini-batch training for stochastic optimization. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 661–670. ACM, 2014. 2

Narasimhan, M. and Bilmes, J. A submodular-supermodular procedure with applications to discriminative structure learning. In *Proceedings of the Twenty-First Conference on Uncertainty in Artificial Intelligence*, pp. 404–412. AUAI Press, 2005. 2, 4, 5

Nemhauser, G. L., Wolsey, L. A., and Fisher, M. L. An analysis of approximations for maximizing submodular set functionsi. *Mathematical programming*, 14(1):265–294, 1978. 2

Podgornaia, A. I. and Laub, M. T. Pervasive degeneracy and epistasis in a protein-protein interface. *Science*, 347(6222):673–677, 2015. 7

Rasmussen, C. E. and Williams, C. K. I. *Gaussian Processes for Machine Learning*. MIT Press, 2006. 2

Romero, P. A., Krause, A., and Arnold, F. H. Navigating the protein fitness landscape with gaussian processes. *Proceedings of the National Academy of Sciences*, 110(3):E193–E201, 2013. 2

Saito, Y., Oikawa, M., Nakazawa, H., Niide, T., Kameda, T., Tsuda, K., and Umetsu, M. Machine-learning-guided mutagenesis for directed evolution of fluorescent proteins. *ACS synthetic biology*, 2018. doi: 10.1021/acssynbio.8b00155. 2

Srinivas, N., Krause, A., Kakade, S., and Seeger, M. Gaussian process optimization in the bandit setting: No regret and experimental design. In *Proc. International Conference on Machine Learning (ICML)*, 2010. 2

Voigt, C. A., Mayo, S. L., Arnold, F. H., and Wang, Z.-G. Compuationally focusing the directed evolution of proteins. *Journal of Cellular Biochemistry Supplement*, 37:58–63, 2001. 1

Wang, Z., Zhou, B., and Jegelka, S. Optimization as estimation with gaussian processes in bandit settings. In *Artificial Intelligence and Statistics*, pp. 1022–1031, 2016. 2

Wu, N. C., Dai, L., Olson, C. A., Lloyd-Smith, J. O., and Sun, R. Adaptation in protein fitness landscapes is facilitated by indirect paths. *Elife*, 5: e16965, 2016. 7

Wu, Z., Kan, S. B. J., Lewis, R. D., Wittmann, B. J., and Arnold, F. H. Machine-learning-assisted

directed protein evolution with combinatorial libraries. *arXiv preprint arXiv:1902.07231*, 2019. 2

Yang, L. and Carbonell, J. Buy-in-bulk active learning. In *Advances in Neural Information Processing Systems*, pp. 2229–2237, 2013. 2

Zinkevich, M., Weimer, M., Li, L., and Smola, A. J. Parallelized stochastic gradient descent. In *Advances in neural information processing systems*, pp. 2595–2603, 2010. 2