Near-linear time Gaussian process optimization with adaptive batching and resparsification

Daniele Calandriello [*1] **Luigi Carratino** [*2] **Alessandro Lazaric** [3] **Michal Valko** [4] **Lorenzo Rosasco** [2 5 6]

Abstract

Gaussian processes (GP) are one of the most successful frameworks to model uncertainty. However, GP optimization (e.g., GP-UCB) suffers from major scalability issues. Experimental time grows linearly with the number of evaluations, unless candidates are selected in batches (e.g., using GP-BUCB) and evaluated in parallel. Furthermore, computational cost is often prohibitive since algorithms such as GP-BUCB require a time at least quadratic in the number of dimensions and iterations to select each batch. In this paper, we introduce BBKB (Batch Budgeted Kernel Bandits), the first no-regret GP optimization algorithm that provably runs in near-linear time and selects candidates in batches. This is obtained with a new guarantee for the tracking of the posterior variances that allows BBKB to choose increasingly larger batches, improving over GP-BUCB. Moreover, we show that the same bound can be used to adaptively delay costly updates to the sparse GP approximation used by BBKB, achieving a near-constant per-step amortized cost. These findings are then confirmed in several experiments, where BBKB is much faster than state-of-the-art methods.

1. Introduction

Gaussian process (GP) optimization is a principled way to optimize a black-box function from noisy evaluations (i.e., sometimes referred to as *bandit* feedback). Due to the presence of noise, the optimization process is modeled as a sequential *learning* problem, where at each step t:
1) the learner chooses candidate $\mathbf{x}_t$ out of a decision set $\mathcal{A}$;
2) the environment evaluates $f(\mathbf{x}_t)$ and returns a noisy feedback y_t to the learner;
3) the learner uses y_t to guide its subsequent choices.

The goal of the learner is to converge over time to a global optimal candidate. This goal is often formalized as a *regret minimization* problem, where the performance of the learner is evaluated by the cumulative value of the candidates chosen over time (i.e., $\sum_t f(\mathbf{x}_t)$) compared to the optimum of the function $f^\star = \max_{\mathbf{x}} f(\mathbf{x})$. While many GP optimization algorithms come with strong theoretical guarantees and are empirically effective, most of them suffer from experimental and/or computational scalability issues.

Experimental scalability. GP optimization algorithms usually follow a sequential interaction protocol, where at each step t, they wait for the feedback y_t before proposing a new candidate $\mathbf{x}_{t+1}$. As such, the *experimentation time* grows linearly with t, which may be impractical in applications where each evaluation may take long time to complete (e.g., in lab experiments). This problem can be mitigated by switching to *batched* algorithms, which at step t propose a batch of candidates that are evaluated in parallel. After the batch is evaluated, the algorithm integrates the feedbacks and move on to select the next batch. This strategy reduces the experimentation time, but it may degrade the optimization performance: the candidates in the batch are picked by a strategy based on less feedback than the one used by a sequentially strategy that selects candidates based on the feedback of each previous choice. Many approaches have been proposed for batched GP optimization. Among those with theoretical guarantees, some are based on sequential greedy selection (Desautels et al., 2014), entropy search (Hennig & Schuler, 2012), determinantal point process sampling (Kathuria et al., 2016), or multi-agent cooperation (Daxberger & Low, 2017), as well as many heuristics for which no regret guarantees (Chevalier & Ginsbourger, 2013; Shah & Ghahramani, 2015). However, they all suffer from the same computational limitations of classical GP methods.

Computational scalability. The computational complexity of choosing a single candidate in classical GP methods grows quadratically with the number of evaluations. This

[*]Equal contribution [1]Istituto Italiano di Tecnologia, Genova, Italy (now at Deepmind, Paris, France) [2]MaLGa - Dibris - Università degli Studi di Genova, Italy [3]Facebook AI Research, Paris, France [4]DeepMind, Paris, France [5]MIT, Cambridge, MA, USA [6]Istituto Italiano di Tecnologia, Genova, Italy. Correspondence to: Daniele Calandriello <daniele.calandriello@iit.it>, Luigi Carratino <luigi.carratino@dibris.unige.it>.

Proceedings of the 37^{th} International Conference on Machine Learning, Online, PMLR 119, 2020.

makes it impractical to optimize complex functions, which require many steps before converging. Many approaches exist to improve scalability. Some have been proposed in the context of sequential GP optimization, such as those based on inducing points and sparse GP approximation (Quinonero-Candela et al., 2007; Calandriello et al., 2019), variational inference (Huggins et al., 2019), random fourier features (Mutny & Krause, 2018), and grid based methods (Wilson & Nickisch, 2015). While some of these methods come with regret and computational guarantees, they rely on a strict sequential protocol, and therefore they are subject to the experimental bottleneck. Other scalable approximations are specific to batched methods, such as Markov approximation (Daxberger & Low, 2017) and Gaussian approximation (Shah & Ghahramani, 2015). However, these methods fail to guarantee either low regret or scalability.

State of the art. Existing GP optimization approaches can be split into Bandit/frequentist approaches (Lattimore & Szepesvári, 2020) and Bayesian approaches (Rasmussen & Williams, 2006), depending on the underlying assumptions on f, or can be considered as heuristics if they do not come with a principled regret analysis. Most methods that come with frequentist regret guarantees, which are the focus of our work, choose candidates using either an upper confidence bound (UCB) or Thompson sampling (TS) approach. We give here an overview of these baselines, and will discuss later how our approach compares with Bayesian approaches, such as those based on expected improvement (Jones et al., 1998) and knowledge gradient (Ryzhov et al., 2012), or heuristic, such as those based on genetic algorithms (Real et al., 2019) and local searches (Choromanski et al., 2019).

GP-UCB (Srinivas et al., 2010) is one of the most popular algorithm for GP optimization and it suffers a regret $\mathcal{O}(\sqrt{T\gamma_T})$, where γ_T is the maximal mutual information gain of a GP after T evaluations (Srinivas et al., 2010; Chowdhury & Gopalan, 2017).[1] Among approximate GP optimization methods, Budgeted Kernelized Bandits (BKB) (Calandriello et al., 2019) and Thompson sampling with quadrature random features (TS-QFF) (Mutny & Krause, 2018) are currently the only provably scalable methods that achieve the $\mathcal{O}(\sqrt{T\gamma_T})$ rate with sub-cubic computational complexity. However they both fail to achieve a fully satisfactory runtime. TS-QFF's complexity[2] $\widetilde{\mathcal{O}}(T2^d d_{\text{eff}}^2)$ scales exponentially in d, and therefore can only be applied to low-dimensional input spaces, while BKB's complexity is still quadratic $\widetilde{\mathcal{O}}(T^2 d_{\text{eff}}^2)$. Furthermore, both TS-QFF and BKB are constrained to a sequential protocol and therefore suffer from poor experimental scalability.

In batch GP optimization, Desautels et al. (2014) introduced a batched version of GP-UCB (GP-BUCB) that can effectively deal with delayed feedback, essentially matching the rate of GP-UCB. Successive methods improved on this approach (Daxberger & Low, 2017, App. G) but are too expensive to scale and/or require strong assumptions on the function f (Contal et al., 2013). Kathuria et al. (2016) uses determinantal point process (DPP) sampling to globally select the batch of points, but DPP sampling is an expensive process in itself, requiring cubic time in the number of alternatives. Although some MCMC-based approximate DPP sampler are scalable, they do not provide sufficiently strong guarantees to prove low regret. Similarly Daxberger & Low (2017) use a Markov-based approximation to select queries, but lose all regret guarantees in the process. In general, the time and space complexity of selecting each candidate $\mathbf{x}_t$ in the batch remains at least $\mathcal{O}(t^2)$, resulting in an overall $\mathcal{O}(T^3)$ time and $\mathcal{O}(T^2)$ space complexity, which is prohibitive beyond a few thousands evaluations.

Contributions. In this paper we introduce a novel sparse approximation for batched GP-UCB, Batch Budgeted Kernelized Bandits (BBKB) with a *constant* $\mathcal{O}(d_{\text{eff}}^2)$ amortized per-step complexity that can easily scale to tens of thousands of iterations. If $\mathcal{A}$ is finite with A candidates, we prove that BBKB runs in near-linear $\widetilde{\mathcal{O}}(TAd_{\text{eff}}^2 + Td_{\text{eff}}^3)$ time and $\widetilde{\mathcal{O}}(Ad_{\text{eff}}^2)$ space and it achieves a regret of order $\mathcal{O}(\sqrt{T\gamma_T})$, thus matching both GP-UCB and GP-BUCB at a fraction of the computational costs (i.e., their complexity scales as $O(T^3)$). This is achieved with two new results of independent interest. First we introduce a new adaptive schedule to select the sizes of the batches of candidates, where the batches selected are larger than the ones used by GP-BUCB (Desautels et al., 2014) while providing the same regret guarantees. Second we prove that the same adaptive schedule can be used to delay updates to BKB's sparse GP approximation (Calandriello et al., 2019), also without compromising regret. This results in large computational savings (i.e., from $\widetilde{\mathcal{O}}(td_{\text{eff}}^2)$ to $\widetilde{\mathcal{O}}(d_{\text{eff}}^2)$ per-step complexity) even in the sequential setting, since updates to the sparse GP approximation, i.e., resparsifications, are the most expensive operation performed by BKB. Delayed resparsifications also allow us to exploit important implementation optimizations in BBKB, such as rank-one updates and lazy updates of GP posterior. We also show that our approach can be combined with existing initialization procedures, both to guarantee a desired minimum level of experimental parallelism and to include pre-existing feedback to bootstrap the optimization problem. We validate our approach on several datasets, showing that BBKB matches or outperforms existing methods in both regret and runtime.

[1] Recently, Calandriello et al. (2019) connected this quantity to the so-called effective dimension d_{eff} of the GP.

[2] The $\widetilde{\mathcal{O}}(\cdot)$ notation ignores logarithmic dependencies.

2. Preliminaries

Setting. A learner is provided with a decision set $\mathcal{A}$ (e.g., a compact set in $\mathbb{R}^d$) and it sequentially selects candidates $\mathbf{x}_1, \dots, \mathbf{x}_T$ from $\mathcal{A}$. At each step t, the learner receives a feedback $y_t \stackrel{\text{def}}{=} f(\mathbf{x}_t) + \eta_t$, where f is an unknown function, and η_t is an additive noise drawn i.i.d. from $\mathcal{N}(0, \xi^2)$.[3] We denote by $\mathbf{X}_t \stackrel{\text{def}}{=} [\mathbf{x}_1, \dots, \mathbf{x}_t]^\intercal \in \mathbb{R}^{t \times d}$ the matrix of the candidates selected so far, and with $\mathbf{y}_t \stackrel{\text{def}}{=} [y_1, \dots, y_t]^\intercal$ the corresponding feedback. We evaluate the performance of the learner by its regret, i.e., $R_T \stackrel{\text{def}}{=} \sum_{t=1}^T f^\star - f(\mathbf{x}_t)$, where $f^\star = \max_{\mathbf{x} \in \mathcal{A}} f(\mathbf{x})$ is the maximum of f. In many applications (e.g., optimization of chemical products) it is possible to execute multiple experiments in parallel. In this case, at step $t = 1$ the learner can select *a batch* of candidates and wait for all feedback $y_1, .., y_{t'}$ before moving to the next batch, starting at $t' > t$. To relate steps with their batch, we denote by $\texttt{fb}(t)$ the index of the last step of the previous batch, i.e., at step t we have access only to feedback $\mathbf{y}_{\texttt{fb}(t)}$ up to step $\texttt{fb}(t)$. Finally, $[t] = \{1, \dots, t\}$ denotes the set of integers up to t.

Sparse Gaussian processes and Nyström embeddings. GPs (Rasmussen & Williams, 2006) are traditionally defined in terms of a mean function μ, which we assume to be zero, and a covariance defined by the (bounded) kernel function $k : \mathcal{A} \times \mathcal{A} \to [0, \kappa^2]$. Given μ, k, and some data, the learner can compute the posterior of the GP.

In the following we introduce the GP posterior using a formulation based on *inducing points* (Quinonero-Candela et al., 2007; Huggins et al., 2019), also known as sparse GP approximations, which is later convenient to illustrate our algorithm. Given a so-called dictionary of inducing points $\mathcal{S} \stackrel{\text{def}}{=} \{\mathbf{x}_i\}_{i=1}^m$, let $\mathbf{K}_\mathcal{S} \in \mathbb{R}^{m \times m}$ be the kernel matrix constructed by evaluating $k(\mathbf{x}_i, \mathbf{x}_j)$ for all the points in $\mathcal{S}$, and similarly let $\mathbf{k}_\mathcal{S}(\mathbf{x}) = [k(\mathbf{x}_1, \mathbf{x}), \dots, k(\mathbf{x}_m, \mathbf{x})]^\intercal$. Then we define a Nyström embedding as $\mathbf{z}(\cdot, \mathcal{S}) \stackrel{\text{def}}{=} \mathbf{K}_\mathcal{S}^{+/2} \mathbf{k}_\mathcal{S}(\cdot) : \mathbb{R}^d \to \mathbb{R}^m$, where $(\cdot)^{+/2}$ indicates the square root of the pseudo-inverse. We can now introduce the matrix $\mathbf{Z}(\mathbf{X}_t, \mathcal{S}) = [\mathbf{z}(\mathbf{x}_1, \mathcal{S}), \dots, \mathbf{z}(\mathbf{x}_t, \mathcal{S})]^\intercal \in \mathbb{R}^{t \times m}$ containing all candidates selected so far after embedding, and define $\mathbf{V}_t = \mathbf{Z}(\mathbf{X}_t, \mathcal{S})^\intercal \mathbf{Z}(\mathbf{X}_t, \mathcal{S}) + \lambda \mathbf{I} \in \mathbb{R}^{m \times m}$. Following Calandriello et al. (2019), the BKB approximation of the posterior mean, covariance, and variance is

$$\widetilde{\mu}_t(\mathbf{x}_i, \mathcal{S}) = \mathbf{z}(\mathbf{x}_i, \mathcal{S})^\intercal \mathbf{V}_t^{-1} \mathbf{Z}(\mathbf{x}_i, \mathcal{S})^\intercal \mathbf{y}_t, \tag{1}$$

$$\widetilde{k}_t(\mathbf{x}_i, \mathbf{x}_j, \mathcal{S}) = \tfrac{1}{\lambda}\Big(k(\mathbf{x}_i, \mathbf{x}_j) - \mathbf{z}(\mathbf{x}_i, \mathcal{S})^\intercal \mathbf{z}(\mathbf{x}_j, \mathcal{S})\Big) + \mathbf{z}(\mathbf{x}_i, \mathcal{S})^\intercal \mathbf{V}_t^{-1} \mathbf{z}(\mathbf{x}_j, \mathcal{S}), \tag{2}$$

$$\widetilde{\sigma}_t^2(\mathbf{x}_i, \mathcal{S}) = \widetilde{k}_t(\mathbf{x}_i, \mathbf{x}_i, \mathcal{S}), \tag{3}$$

where λ is a parameter to be tuned. The subscript t in $\widetilde{\mu}_t$ and $\widetilde{\sigma}_t$ indicates what we already observed (i.e., $\mathbf{X}_t$ and $\mathbf{y}_t$), and $\mathcal{S}$ indicates the dictionary used for the embedding. Moreover, if $\mathcal{S}_{\text{exact}}$ is a *perfect* dictionary we recover a formulation almost equivalent[4] to the standard posterior mean and covariance of a GP, which we denote with $\mu_t(\mathbf{x}) \stackrel{\text{def}}{=} \widetilde{\mu}_t(\mathbf{x}, \mathcal{S}_{\text{exact}})$ and $\sigma_t^2(\mathbf{x}) \stackrel{\text{def}}{=} \widetilde{\sigma}_t^2(\mathbf{x}, \mathcal{S}_{\text{exact}})$. Examples of possible $\mathcal{S}_{\text{exact}}$ are the whole set $\mathcal{A}$ if finite, or the set of all candidates $\{\mathbf{x}_1, \dots, \mathbf{x}_T\}$ selected so far.

Finally, we define the effective dimension after t steps as

$$d_{\text{eff}}(\mathbf{X}_t) = \textstyle\sum_{s=1}^t \sigma_t^2(\mathbf{x}_s) = \operatorname{Tr}(\mathbf{K}_t(\mathbf{K}_t + \lambda \mathbf{I})^{-1}).$$

Intuitively, $d_{\text{eff}}(\mathbf{X}_t)$ captures the effective number of parameters in f, i.e., the posterior f can be represented using roughly $d_{\text{eff}}(\mathbf{X}_t)$ coefficients. We use d_{eff} to denote $d_{\text{eff}}(\mathbf{X}_T)$ at the end of the process. Note that d_{eff} is equivalent to the maximal conditional mutual information γ_T of the GP (Srinivas et al., 2010), up to logarithmic terms (Calandriello et al., 2017a, Lem. 1).

The GP-UCB family. GP-UCB-based algorithms aim to construct an *acquisition function* $u_t(\cdot) : \mathcal{A} \to \mathbb{R}$ to act as an upper confidence bound (UCB) for the unknown function f. Whenever $u_t(\mathbf{x})$ is a valid UCB (i.e., $f(\mathbf{x}) \leq u_t(\mathbf{x})$) and it converges to $f(\mathbf{x})$ "sufficiently" fast, selecting candidates that are optimal w.r.t. to u_t leads to low regret, i.e., the value $f(\mathbf{x}_{t+1})$ of $\mathbf{x}_{t+1} = \arg\max_{\mathbf{x} \in \mathcal{A}} u_t(\mathbf{x})$ tends to $\max_{\mathbf{x} \in \mathcal{A}} f(\mathbf{x})$ as t increases.

In particular, GP-UCB (Srinivas et al., 2010) defines $u_t(\mathbf{x}) = \mu_t(\mathbf{x}) + \beta_t \sigma_t(\mathbf{x})$, for a properly chosen $\beta_t \in \mathbb{R}_+$. Unfortunately, GP-UCB is computationally and experimentally inefficient, as evaluating $u_t(\mathbf{x})$ requires $\mathcal{O}(t^2)$ per-step and no parallel experiments are possible. To improve computations, BKB (Calandriello et al., 2019), for a specific $\widetilde{\beta}_t \in \mathbb{R}_+$, replaces u_t with an approximate $\widetilde{u}_t^{\text{BKB}}(\mathbf{x}) = \widetilde{\mu}_t(\mathbf{x}, \mathcal{S}_t) + \widetilde{\beta}_t \widetilde{\sigma}_t(\mathbf{x}, \mathcal{S}_t)$, which is proven to be sufficiently close to u_t to achieve low regret. However, maintaining accuracy requires $\mathcal{O}(t)$ per step to update the dictionary $\mathcal{S}_t$ at each iteration, and the queries are still selected sequentially. GP-BUCB (Desautels et al., 2014) tries to increase GP-UCB's experimental efficiency by selecting a batch of queries that are all evaluated in parallel. In particular, for a certain $\alpha_t \in \mathbb{R}_+$, GP-BUCB approximate the UCB as $\widetilde{u}_t^{\text{GP-BUCB}}(\mathbf{x}) = \mu_{\texttt{fb}(t)}(\mathbf{x}) + \alpha_t \sigma_t(\mathbf{x})$, where the mean is not updated until new feedback arrives, while due to its definition the variance only depends on $\mathbf{X}_t$ and can be updated in an unsupervised manner. Nonetheless,

[3] Candidates are sometime referred to as actions, arms, or queries, and feedback is sometimes called reward or observation.

[4] We refer to $\widetilde{\mu}_t$ and $\widetilde{\sigma}_t^2$ as posteriors with a slight abuse of terminology. In particular, up to a $1/\lambda$ rescaling, they correspond to the Bayesian DTC approximation (Quinonero-Candela et al., 2007), which is not a GP posterior in a strictly Bayesian sense. Our rescaling $1/\lambda$ is also not present when deriving the exact μ_t and σ_t^2 as Bayesian posteriors, but is necessary to simplify the notation of our frequentist analysis. For more details, see Appendix A.

GP-BUCB is as computationally slow as GP-UCB. More details about these methods are reported in Appendix A.

Controlling regret in batched Bayesian optimization. For all steps *within* a batch, GP-BUCB can be seen as *fantasizing* or *hallucinating* a constant feedback $\mu_{\mathtt{fb}(t)}(\mathbf{x}_t)$ so that the mean does not change, while the variances keep *shrinking*, thus promoting diversity in the batch. However, incorporating fantasized feedback causes $u_t^{\text{GP-BUCB}}$ to drift away from u_t to the extent that it may not be a valid UCB anymore. Desautels et al. (2014) show that this issue can be managed by adjusting GP-UCB's parameter β_t. In fact, it is possible to take the $u_t^{\text{GP-BUCB}}$ at the beginning of the batch (which is a valid UCB by definition), and *correct* it to hold for each hallucinated step as $f(\mathbf{x}) \leq \mu_{\mathtt{fb}(t)}(\mathbf{x}) + \rho_{\mathtt{fb}(t),t}(\mathbf{x})\beta_{\mathtt{fb}(t)}\sigma_t(\mathbf{x})$, where $\rho_{\mathtt{fb}(t),t}(\mathbf{x}) \stackrel{\text{def}}{=} \frac{\sigma_{\mathtt{fb}(t)}(\mathbf{x})}{\sigma_t(\mathbf{x})}$ is the posterior standard deviation ratio. By using any $\alpha_t \geq \rho_{\mathtt{fb}(t),t}(\mathbf{x})\beta_{\mathtt{fb}(t)}$, we have that $u_t^{\text{GP-BUCB}}$ is a valid UCB. As the length of the batch increases, the ratio $\rho_{\mathtt{fb}(t),t}$ may become larger, and the UCB becomes less and less tight. Crucially, the drift of the ratio can be estimated.

Proposition 1 (Desautels et al. (2014), Prop. 1)**.** *At any step t, for any $\mathbf{x} \in \mathcal{A}$ the posterior ratio is bounded as*

$$\rho_{\mathtt{fb}(t),t}(\mathbf{x}) \stackrel{\text{def}}{=} \frac{\sigma_{\mathtt{fb}(t)}(\mathbf{x})}{\sigma_t(\mathbf{x})} \leq \prod_{s=\mathtt{fb}(t)+1}^{t} \left(1 + \sigma^2_{s-1}(\mathbf{x}_s)\right).$$

Based on this result, GP-BUCB continues the construction of the batch while $\prod_{s=\mathtt{fb}(t)+1}^{t} \left(1 + \sigma^2_{s-1}(\mathbf{x}_s)\right) \leq C$ for some designer-defined threshold of drift C. Therefore, applying Proposition 1, we have that the ratio $\rho_{\mathtt{fb}(t),t}(\mathbf{x}) \leq C$ for any $\mathbf{x}$, and setting $\alpha_t \stackrel{\text{def}}{=} C\beta_{\mathtt{fb}(t)}$ guarantees the validity of the UCB, just as in GP-UCB. As a consequence, GP-UCB's analysis can be leveraged to provide guarantees on the regret of GP-BUCB.

3. Efficient Batch GP Optimization

In this section, we introduce BBKB which both generalizes and improves over GP-BUCB and BKB.

3.1. The algorithm

The pseudocode of BBKB is presented in Algorithm 1. The dictionary is initially empty, and we have $\widetilde{\mu}_0(\mathbf{x}) = 0$ and $\widetilde{\sigma}_0(\mathbf{x}, \{\}) = k(\mathbf{x}, \mathbf{x})/\lambda = \sigma_0(\mathbf{x})$. At each step t BBKB chooses the next candidate $\mathbf{x}_{t+1}$ as the maximizer of the UCB $\widetilde{u}_t(\mathbf{x}) = \widetilde{\mu}_{\mathtt{fb}(t)}(\mathbf{x}, \mathcal{S}_{\mathtt{fb}(t)}) + \widetilde{\alpha}_{\mathtt{fb}(t)}\widetilde{\sigma}_t(\mathbf{x}, \mathcal{S}_{\mathtt{fb}(t)})$, which combines BKB and GP-BUCB's approaches with a new element. In $\widetilde{u}_t$, not only we delay feedback updates as we use the posterior mean computed at the end of the last batch $\widetilde{\mu}_{\mathtt{fb}(t)}$ but, unlike BKB, we keep using the same dictionary $\mathcal{S}_{\mathtt{fb}(t)}$ for all steps in a batch. While freezing the dictionary leads to significantly reducing the computational complexity, delaying feedback and dictionary updates may result in poor UCB approximation. Similar to GP-BUCB, after selecting $\mathbf{x}_{t+1}$ we test the condition in (L5) to decide whether to continue the batch, a *batch construction step*, or not, a *resparsification step*. The specific formulation of our condition is crucial to guarantee near-linear runtime, and improves over the condition used in GP-BUCB. Notice that if we update dictionary and feedback at each step, BBKB reduces to BKB (up to an improved $\widetilde{\alpha}_t$ as discussed in the next section), while if $\mathcal{S}_t = \mathbf{X}_t$ we recover GP-BUCB, but with an improved terminating rule for batches.

Algorithm 1 BBKB

Require: Set of candidates $\mathcal{A}$, $\{\widetilde{\alpha}_t\}_{t=1}^T$, T, $\widetilde{C}$, $\{\overline{q}_t\}_{t=1}^T$

1: Sample $\mathbf{x}_1$ uniformly, receive $\mathbf{y}_1$
2: Initialize $\mathcal{S}_0 = \{\}$, $\mathtt{fb}(0) = 0$
3: **for** $t = \{0, \dots, T-1\}$ **do**
4: Select $\mathbf{x}_{t+1} = \arg\max_{\mathbf{x}\in\mathcal{A}} \widetilde{u}_t(\mathbf{x}, \mathcal{S}_{\mathtt{fb}(t)})$
5: **if** $1 + \sum_{s=\mathtt{fb}(t)+1}^{t+1} \widetilde{\sigma}^2_{\mathtt{fb}(t)}(\mathbf{x}_s, \mathcal{S}_{\mathtt{fb}(t)}) \leq \widetilde{C}$ **then**
6: // $\mathtt{fb}(t+1) = \mathtt{fb}(t)$, *batch construction step*
7: Update $\widetilde{u}_{t+1}(\mathbf{x}_{t+1}, \mathcal{S}_{\mathtt{fb}(t+1)})$ with the new $\widetilde{\sigma}_{t+1}$
8: Update $\widetilde{u}_{t+1}(\mathbf{x}_i, \mathcal{S}_{\mathtt{fb}(t+1)})$ for all $\{\mathbf{x} : \widetilde{u}_t(\mathbf{x}, \mathcal{S}_{\mathtt{fb}(t)}) \geq \widetilde{u}_{t+1}(\mathbf{x}_{t+1}, \mathcal{S}_{\mathtt{fb}(t)})\}$
9: **else**
10: // $\mathtt{fb}(t+1) = t+1$, *resparsification step*
11: Initialize $\mathcal{S}_{\mathtt{fb}(t+1)} = \emptyset$
12: **for** $\mathbf{x}_s \in \mathbf{X}_{\mathtt{fb}(t+1)}$ **do**
13: Set $\widetilde{p}_{\mathtt{fb}(t+1),s} = \overline{q}_t \cdot \widetilde{\sigma}^2_{\mathtt{fb}(t)}(\mathbf{x}_s)$
14: Draw $z_{\mathtt{fb}(t+1),s} \sim Bernoulli(\widetilde{p}_{\mathtt{fb}(t+1),s})$
15: If $z_{\mathtt{fb}(t+1),s} = 1$, add $\mathbf{x}_s$ in $\mathcal{S}_{\mathtt{fb}(t+1)}$
16: **end for**
17: Get feedback $\{y_s\}_{s=\mathtt{fb}(t)+1}^{\mathtt{fb}(t+1)}$
18: Update $\widetilde{\mu}_{\mathtt{fb}(t+1)}$ and $\widetilde{\sigma}_{\mathtt{fb}(t+1)}$ for all $\mathbf{x} \in \mathcal{A}$
19: **end if**
20: **end for**

In a batch construction step, we keep using the same dictionary in computing the UCBs used to select the next candidate. On the other hand, if condition (L5) determines that UCBs may become too loose, we interrupt the batch and update the sparse GP approximation, i.e., we resparsify the dictionary. To do this we employ BKB's posterior sampling procedure in L11-16. For each candidate $\mathbf{x}_s$ selected so far, we compute an inclusion probability $\widetilde{p}_{\mathtt{fb}(t+1),s} = \overline{q}_t \cdot \widetilde{\sigma}_{\mathtt{fb}(t)}(\mathbf{x}_s)$, where $\overline{q}_t \geq 1$ is a parameter trading-off the size of $\mathcal{S}$ and the accuracy of the approximations, and we add $\mathbf{x}_s$ to the new dictionary with probability $\widetilde{p}_{\mathtt{fb}(t+1),s}$. A crucial difference w.r.t. BKB is that in computing the inclusion probability we use the posterior variances computed at the beginning of the batch (instead of $\widetilde{\sigma}_{\mathtt{fb}(t+1)}$). While this introduces a further source of approximation, in the next section we show that this error can be controlled. The resulting dictionary is then used to compute the embedding $\mathbf{z}_{\mathtt{fb}(t+1)}$ and the UCB values whenever needed.

Maximizing the UCB. Since in general u_t is a highly nonlinear, non-convex function, it may be NP-hard to compute $\mathbf{x}_{t+1}$ as its $\arg\max$ over $\mathcal{A}$. To simplify the exposition, in

the rest of the paper we assume that $\mathcal{A}$ is finite with cardinality A such that simple enumeration of all candidates in $\mathcal{A}$ is sufficient to exactly optimize the UCB. Both this assumption and the runtime dependency on A can be easily removed if an efficient way to optimize u_t over $\mathcal{A}$ is provided (e.g., see Mutny & Krause (2018) for the special case of $d = 1$ or when k is an additive kernel).

Moreover, when $\mathcal{A}$ is finite BBKB can be implemented much more efficiently. In particular, many of the quantities used by BBKB can be precomputed once at the beginning of the batch, such as pre-embedding all arms. In addition keeping the embeddings fixed during the batch allows us to update the posterior variances using efficient rank-one updates, combining the efficiency of a parametric method with the flexibility of non-parametric GPs. Finally, when both dictionary and feedback are fixed we can leverage *lazy* covariance evaluations, which allows us to exactly compute the $\mathbf{x}_{t+1}$ while only updating a small fraction of the UCBs (see Appendix A for more details).

3.2. Computational analysis

The global runtime of BBKB is $\mathcal{O}\big(TAm^2 + BTm + B(Am^2 + m^3)\big)$, where $m = \max_t |\mathcal{S}_{\mathtt{fb}(t)}|$ is the maximum size of the dictionary/embedding across batches, and B the number of batches/resparsifications (see Appendix C for details). In order to obtain a near-linear runtime, we need to show that both $|\mathcal{S}_t|$ and B are nearly-constant.

Theorem 1. *Given $\delta \in (0,1)$, $1 \leq \widetilde{C}$, and $1 \leq \lambda$, run* BBKB *with $\overline{q}_t \geq 8\log(4t/\delta)$. Then, w.p. $1-\delta$*

1) *For all $t \in [T]$ we have $|\mathcal{S}_t| \leq 9\widetilde{C}(1+\kappa^2/\lambda)\overline{q}_t d_{\mathit{eff}}(\mathbf{X}_t)$.*
2) *Moreover, the total number of resparsification B performed by* BBKB *is at most $\mathcal{O}(d_{\mathit{eff}}(\mathbf{X}_t))$.*
3) *As a consequence,* BBKB *runs in near-linear time $\widetilde{\mathcal{O}}(TAd_{\mathit{eff}}(\mathbf{X}_t)^2)$.*

Theorem 1 guarantees that whenever d_{eff}, or equivalently γ_T, is near-constant (i.e., $\widetilde{\mathcal{O}}(1)$), BBKB runs in $\widetilde{\mathcal{O}}(TA)$. Srinivas et al. (2010) shows that this is the case for common kernels, e.g., $\gamma_T \leq \mathcal{O}(d\log(T))$ for the linear kernel and $\gamma_T \leq \mathcal{O}(\log(T)^d)$ for the Gaussian kernel.

Among sequential GP-Opt algorithms, BBKB is not only much faster than the original GP-UCB $\widetilde{\mathcal{O}}(T^3A)$ runtime, but also much faster when compared to BKB's quadratic $\widetilde{\mathcal{O}}(T\max\{A,T\}d_{\text{eff}}^2)$. BBKB's runtime also improves over GP-optimization algorithms that are specialized for stationary kernels (e.g. Gaussian), such as QFF-TS's (Mutny & Krause, 2018) $\widetilde{\mathcal{O}}(TA2^d d_{\text{eff}}^2)$ runtime, without making any assumption on the kernel and without an exponential dependencies on the input dimension d. When compared to batch algorithms, such as GP-BUCB, the improvement is even sharper as all existing batch algorithms that are provably no-regret (Desautels et al., 2014; Contal et al., 2013; Shah & Ghahramani, 2015) share GP-UCB's $\widetilde{\mathcal{O}}(AT^3)$ runtime.

One of the central elements of this result is BBKB's adaptive batch terminating condition. As a comparison, GP-BUCB uses $\prod_{s=\mathtt{fb}(t)+1}^{t+1}(1+\sigma^2_{\mathtt{fb}(t)}(\mathbf{x}_s))$ as a batch termination condition, but due to Weierstrass product inequality

$$1+\textstyle\sum_{s=\mathtt{fb}(t)+1}^{t+1}\sigma^2_{\mathtt{fb}(t)}(\mathbf{x}_s) \leq \prod_{s=\mathtt{fb}(t)+1}^{t+1}(1+\sigma^2_{\mathtt{fb}(t)}(\mathbf{x}_s)),$$

and the product is always larger than the sum which BBKB uses. Thanks to the tighter bound, we obtain larger batches and can guarantee that at most $\widetilde{\mathcal{O}}(d_{\text{eff}})$ batches are necessary over T steps. This implies that, unless $d_{\text{eff}} \to \infty$ in which case the optimization would not converge in the first place, the size of the batches must on average grow linearly with T to compensate. In addition to this guarantee on the average batch size, in the next section we show how smarter initialization schemes can guarantee a minimum batch size, which is useful to fully utilize any desired level of parallelism. Finally, note that the A factor reported in the runtime is pessimistic, since BBKB recomputes only a small fraction of UCB's at each step thanks to lazy evaluations, and should be considered simply as a proxy of the time required to find the UCB maximizer.

3.3. Regret analysis

We report regret guarantees for BBKB in the so-called *frequentist* setting. While the algorithm uses GP tools to define and manage the uncertainty in estimating the unknown function f, the analysis of BBKB does not rely on any *Bayesian* assumption about f being actually drawn from the prior $\mathrm{GP}(0,k)$, and it only requires f to be bounded in the norm associated to the RKHS induced by the kernel function k.

Theorem 2. *Assume $\|f\|_{\mathcal{H}} \leq F < \infty$, and let ξ^2 be the variance of the noise η_t. For any desired, $0 < \delta < 1$, $1 \leq \lambda$, $1 \leq \widetilde{C}$, if we run* BBKB *with $\overline{q}_t \geq 72\widetilde{C}\log(4t/\delta)$, $\widetilde{\alpha}_{\mathtt{fb}(t)} = \widetilde{C}\widetilde{\beta}_{\mathtt{fb}(t)}$, and*

$$\widetilde{\beta}_{\mathtt{fb}(t)} = 2\xi\sqrt{\sum_{s=1}^{\mathtt{fb}(t)}\log\Big(1+3\widetilde{\sigma}^2_{\mathtt{fb}(s-1)}(\mathbf{x}_s)\Big)+\log\big(\tfrac{1}{\delta}\big)} + (1+\sqrt{2})\sqrt{\lambda}F,$$

then, with prob. $1-\delta$, BBKB*'s regret is bounded as*

$$R_T^{\text{BBKB}} \leq 55\widetilde{C}^2 R_T^{\text{GP-BUCB}} \leq 55\widetilde{C}^3 R_T^{\text{GP-UCB}}$$

with $R_T^{\text{GP-UCB}}$ bounded by

$$\sqrt{T}\left(\xi\left(\sum_{t=1}^{T}\sigma^2_{t-1}(\mathbf{x}_t)+\log\big(\tfrac{1}{\delta}\big)\right)+\sqrt{\lambda F^2\sum_{t=1}^{T}\sigma^2_{t-1}(\mathbf{x}_t)}\right).$$

Theorem 2 shows that BBKB achieves essentially the same regret as GP-BUCB and GP-UCB, but at a fraction of the computational cost. Note that $\sum_{t=1}^{T}\sigma^2_{t-1}(\mathbf{x}_t) \approx$

$\log\det(\mathbf{K}_T/\lambda+\mathbf{I}) \approx \gamma_T$ (Srinivas et al., 2010, Lem. 5.4). Such a tight bound is achieved thanks in part to a new confidence interval radius $\widetilde{\beta}_t$. In particular Calandriello et al. (2019) contains an extra $\log\det(\mathbf{K}_T/\lambda+\mathbf{I}) \leq d_{\text{eff}}(\lambda,\mathbf{X}_T)\log(T)$ bounding step that we do not have to make. While in the worst case this is only a $\log(T)$ improvement, empirically the data adaptive bound seems to lead to much better regret.

Discussion. BBKB directly generalizes and improves both BKB and GP-BUCB. If $\widetilde{C}=1$, BBKB is equivalent to BKB, with a improved $\widetilde{\beta}_t$ and a slightly better regret by a factor $\log(T)$, and if $\mathcal{S}_t=\mathcal{S}_{\text{exact}}$ we recover GP-BUCB, with an improved batch termination rule. BBKB's algorithmic derivation and analysis require several new tools. A direct extension of BKB to the batched setting would achieve low regret but be computationally expensive. In particular, it is easy to extend BKB's analysis to delayed feedback, but only if BKB adapts the embedding space (i.e., resparsifies the GP) after every batch construction step to maintain guarantees at all times, i.e., Theorem 1 must hold at all steps and not only at $\texttt{fb}(t)$. However this causes large computation issues, as embedding the points is BKB's most expensive operation, and prevents any kind of lazy evaluation of the UCBs. BBKB solves these two issues with a simple algorithmic fix by *freezing the dictionary* during the batch. However, this bring additional challenges for the analysis. The reason is that while the dictionary is frozen, we may encounter a point $\mathbf{x}_t$ that *cannot be well represented* with the current S_t, but we cannot add $\mathbf{x}_t$ to it since the dictionary is frozen. This requires studying how posterior mean and variance drift away from their values at the beginning of the batch. We tackle this problem by *simultaneously* freezing the dictionary and batching candidates. As we will see in the next section, and prove in the appendix, freezing the dictionary allows us to control the ratio $\widetilde{\sigma}_{\texttt{fb}(t)}(\mathbf{x}_i,\mathcal{S}_{\texttt{fb}(t)})/\widetilde{\sigma}_t(\mathbf{x}_i,\mathcal{S}_{\texttt{fb}(t)})$, obtaining a generalization of Proposition 5. However, changing the posterior mean $\widetilde{\mu}_t(\cdot,\mathcal{S}_{\texttt{fb}(t)})$ without changing the dictionary could still invalidate the UCBs. Batching candidates allows BBKB to continue using the posterior mean $\widetilde{\mu}_{\texttt{fb}(t)}(\cdot,\mathcal{S}_{\texttt{fb}(t)})$, which is known to be accurate, and resolve this issue. By terminating the batches when exceeding a prescribed potential error threshold (i.e., $\widetilde{C}$) we can ignore the intermediate estimate and reconnect all UCBs with the accurate UCBs at the beginning of the batch. This requires a deterministic, worst-case analysis of both the evolution of $\widetilde{\sigma}_t(\mathbf{x}_i)$ and $\sigma_t(\mathbf{x}_i)$, which we provide in the appendix.

Proof sketch. One of the central elements in BBKB's computational and regret analysis is the new adaptive batch terminating condition. In particular, remember that GP-BUCB's regret analysis was centered around the fact that the posterior ratio $\rho_{\texttt{fb}(t),t}(\mathbf{x})$ from Proposition 1 could be controlled using Desautels et al. (2014)'s batch termination rule. However, this result cannot be transferred directly to BBKB for several reasons. First, we must not only control the ratio $\rho_{\texttt{fb}(t),t}(\mathbf{x})$, but also the approximate ratio $\widetilde{\rho}_{\texttt{fb}(t),t}(\mathbf{x},\mathcal{S}) \overset{\text{def}}{=} \frac{\widetilde{\sigma}_{\texttt{fb}(t)}(\mathbf{x},\mathcal{S})}{\widetilde{\sigma}_t(\mathbf{x},\mathcal{S})}$ for some dictionary $\mathcal{S}$, since we are basing most of our choices on $\widetilde{\sigma}_t$ but will be judged based on σ_t (i.e., the real function is based on k and σ_t, not on some $\widetilde{k}$ and $\widetilde{\sigma}_t$). Therefore our termination rule must provide guarantees for both. Second, GP-BUCB's rule is not only expensive to compute, but also hard to approximate. In particular, if we approximated $\sigma_{\texttt{fb}(t)}(\mathbf{x}_s)$ with $\widetilde{\sigma}_{\texttt{fb}(t)}(\mathbf{x}_s)$ in Proposition 1, any approximation error incurred would be compounded multiplicatively by the product resulting in an overall error *exponential* in the length of the batch. Instead, the following novel ratio bound involves only summations.

Lemma 1. *For any kernel k, dictionary $\mathcal{S}$, set of points $\mathbf{X}_t$, $\mathbf{x}\in\mathcal{A}$, and $\texttt{fb}(t)<t$, we have*

$$\widetilde{\rho}_{\texttt{fb}(t),t}(\mathbf{x},\mathcal{S}) \leq 1+\sum\nolimits_{s=\texttt{fb}(t)}^{t}\widetilde{\sigma}^2_{\texttt{fb}(t)}(\mathbf{x},\mathcal{S}).$$

The proof, reported in the appendix, is based only on linear algebra and does not involve any GP-specific derivation, making it applicable to the DTC approximation used by BBKB. Most importantly, it holds regardless of the dictionary $\mathcal{S}$ used (as long as it stays constant) and regardless of which candidates an algorithm might include in the batch. If we replace $\widetilde{\sigma}_{\texttt{fb}(t)}$ with $\sigma_{\texttt{fb}(t)}$ the bound can also be applied to $\rho_{\texttt{fb}(t),t}(\mathbf{x},\mathcal{S})$, giving us an improved version of Proposition 1 as a corollary. Finally, replacing the product in Proposition 1 with the summation in Lemma 1 makes it much easier to analyse it, leveraging this result adapted from (Calandriello et al., 2019).

Lemma 2. *Under the same conditions as Theorem 1, w.p. $1-\delta$, $\forall\ \texttt{fb}(t)\in[T]$ and $\forall\ \mathbf{x}\in\mathcal{A}$ we have*

$$\sigma^2_{\texttt{fb}(t)}(\mathbf{x})/3 \leq \widetilde{\sigma}^2_{\texttt{fb}(t)}(\mathbf{x},\mathcal{S}_{\texttt{fb}(t)}) \leq 3\sigma^2_{\texttt{fb}(t)}(\mathbf{x}).$$

Lemma 2 shows that at the beginning of each batch BBKB, similarly to BKB, does not underestimate uncertainty, i.e., unlike existing approximate batched methods it does not suffer from variance starvation (Wang et al., 2018), Applying Lemma 2 to Lemma 1 we show that our batch terminating condition can provide guarantees on both the approximate ratio $\widetilde{\sigma}_{\texttt{fb}(t)}/\widetilde{\sigma}_t \leq \widetilde{C}$, as well as the exact posterior ratio $\sigma_{\texttt{fb}(t)}/\sigma_t \leq 3\widetilde{C}$ paying only an extra constant approximation factor. Both of these conditions are necessary to obtain the final regret bound.

4. Extensions

In this section we discuss two important extensions of BBKB: **1)** how to leverage initialization to improve experimental parallelism and accuracy, **2)** how to further trade-off a small amount of extra computation to improve parallelism.

Initialization to guarantee minimum batch size. In many cases it is desirable to achieve at least a certain prescribed level of parallelism P, e.g., to be able to fully utilize a server farm with P machines or a lab with P analysis machines[5]. However, BBKB's batch termination rule is designed only to control the ratio error, and might generate batches smaller than P, especially in the beginning when posterior variances are large and their sum can quickly reach the threshold $\widetilde{C}$. However, it is easy to see that if at step $\texttt{fb}(t)$ we have $\max_{\mathbf{x}\in\mathcal{A}} \widetilde{\sigma}^2_{\texttt{fb}(t)}(\mathbf{x}) \leq 1/P$ for all $\mathbf{x}$, then the batch will be at least as large as P.

The same problem of controlling the maximum posterior variance of a GP was studied by Desautels et al. (2014), who showed that a specific initialization scheme (see Appendix A for details) called uncertainty sampling (US) can guarantee that after T_{init} initialization samples, we have that $\max_{\mathbf{x}\in\mathcal{A}} \sigma^2_{T_{\text{init}}}(\mathbf{x}) \leq \gamma_{T_{\text{init}}}/T_{\text{init}}$. Since it is known that for many covariances k the maximum information gain γ_t grows sub-linearly in t, we have that $\gamma_{T_{\text{init}}}/T_{\text{init}}$ eventually reaches the desired $1/P$. For example, for the linear kernel $T_{\text{init}} \leq Pd\log(P)$ suffices, and $T_{\text{init}} \leq \log(P)^{d+1}$ for the Gaussian kernel. All of these guarantees can be transferred to our approximate setting thanks to Lemma 2 and to the monotonicity of σ_t. In particular, after a sufficient number T_{init} of steps of US, and for any $\texttt{fb}(t) > T_{\text{init}}$ we have that

$$\widetilde{\sigma}^2_{\texttt{fb}(t)}(\mathbf{x}, \mathcal{S}_{\texttt{fb}(t)}) \leq 3\sigma^2_{\texttt{fb}(t)}(\mathbf{x}) \leq 3\sigma^2_{T_{\text{init}}}(\mathbf{x}) \leq 3/P,$$

and US can be used to control BBKB's batch size as well.

Initialization to leverage existing data. In many domains GP-optimization is applied to existing problems in hope to improve performance over an existing decision system (e.g., replace uniform exploration or A/B testing with a more sophisticated alternative). In this case, existing historical data can be used to initialize the GP model and improve regret, as it is essentially "free" exploration. However this still present a computational challenge, since computing the GP posterior scales with the number of total evaluations, which includes the initialization. In this aspect, BBKB can be seamlessly integrated with initialization using pre-existing data. All that is necessary is to pre-compute a provably accurate dictionary $\mathcal{S}_{T_{\text{init}}}$ using any batch sampling technique that provides guarantees equivalent to those of Lemma 2, see e.g.,Calandriello et al. (2017a); Rudi et al. (2018). The algorithm then continues as normal from step $T_{\text{init}}+1$ using the embeddings based on $\mathcal{S}_{T_{\text{init}}}$, maintaining all computational and regret guarantees.

Local control of posterior ratios. Finally, we want to highlight that the termination rule of BBKB is just one of many possible rules to guarantee that the posterior ratio is controlled. In particular, while BBKB's rule improves over GP-BUCB's, it is still not optimal. For example, one could imagine recomputing *all* posterior variances at each step and check that $\max_{\mathbf{x}\in\mathcal{A}} \widetilde{\sigma}^2_{\texttt{fb}(t)}(\mathbf{x}, \mathcal{S}_{\texttt{fb}(t)})/\widetilde{\sigma}^2_t(\mathbf{x}, \mathcal{S}_{\texttt{fb}(t)}) \leq \widetilde{C}$.

However this kind of local (i.e., specific to a $\mathbf{x}$) test is computationally expensive, as it requires a sweep over $\mathcal{A}$ and at least $\mathcal{O}(|\mathcal{S}_{\texttt{fb}(t)}|^2)$ time to compute each variance, which is why BBKB and GP-BUCB's termination rule use only global information. To try to combine the best of both worlds, we propose a novel efficient local termination rule.

Lemma 3. *For any kernel k, dictionary $\mathcal{S}$, set of points $\mathbf{X}_t$, $\mathbf{x} \in \mathcal{A}$, and $\texttt{fb}(t) < t$,*

$$\widetilde{\rho}_{\texttt{fb}(t),t}(\mathbf{x}, \mathcal{S}) \leq 1 + \frac{\sum_{s=\texttt{fb}(t)}^{t} \widetilde{k}^2_{\texttt{fb}(t)}(\mathbf{x}, \mathbf{x}_s, \mathcal{S}_{\texttt{fb}(t)})}{\widetilde{\sigma}^2_{\texttt{fb}(t)}(\mathbf{x}, \mathcal{S}_{\texttt{fb}(t)})}$$

Note that $\widetilde{k}^2_{\texttt{fb}(t)}(\mathbf{x}, \mathbf{x}_s) \leq \widetilde{\sigma}^2_{\texttt{fb}(t)}(\mathbf{x})\widetilde{\sigma}^2_{\texttt{fb}(t)}(\mathbf{x}_s)$, due to Cauchy-Bunyakovsky-Schwarz, and therefore this termination rule is tighter than the one in Lemma 1. Moreover, with an argument similar to the one in Lemma 1 we can again show that the termination provably controls both the ratio of exact and approximate posteriors.

Computationally, after a $\widetilde{\mathcal{O}}(d^2_{\text{eff}})$ cost to update $\mathbf{V}_t^{-1}$, computing multiple $\widetilde{k}_{\texttt{fb}(t)}(\mathbf{x}, \mathbf{x}_s, \mathcal{S}_{\texttt{fb}(t)})$ for a fixed $\mathbf{x}_s$ requires only $\widetilde{\mathcal{O}}(d_{\text{eff}})$ time, i.e., it requires only a vector-vector multiplication in the embedded space. Therefore, the total cost of updating the posterior ratio estimates using Lemma 3 is $\widetilde{\mathcal{O}}(Ad_{\text{eff}} + d^2_{\text{eff}})$, while recomputing all variances requires $\mathcal{O}(Ad^2_{\text{eff}})$. However, it still requires a full sweep over all candidates introducing a dependency on A. As commented in the case of posterior maximization, lazy updates can be used to empirically alleviate this dependency. Finally, it is possible to combine both bounds: at first use the global bound from Lemma 1, and then switch to the more computationally expensive local bound of Lemma 3 only if the constructed batch is not "large enough".

5. Experiments

In this section we empirically study the performance in regret and computational costs of BBKB compared to Eps-Greedy, GP-UCB, GP-BUCB, BKB, batch Thompson sampling (ASYNC-TS) (Kandasamy et al., 2018) and genetic algorithms (REG-EVOLUTION) (Real et al., 2019). For BBKB we use both the batch stopping rules presented in Lemma 1 and Lemma 3 calling the two versions Global-BBKB and GlobalLocal-BBKB respectively[6]. For each experimental result we report mean and 95% confidence interval using 10 repetitions. The experiments are implemented in `python` using the `numpy`, `scikit-learn` and `botorch` library, and run on a 16 core dual-CPU server

[5]For simplicity here we assume that all evaluations require the same time and that batch sizes are a multiple of P. This can be easily relaxed at the only expense of a more complex notation.

[6]Code can be found at *github.com/luigicarratino/batch-bkb*

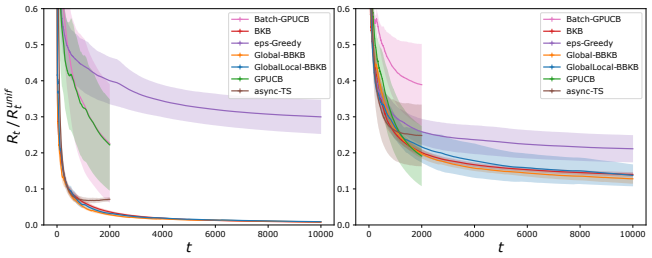

Figure 1: Regret-ratio on Abalone (left) and Cadata (right)

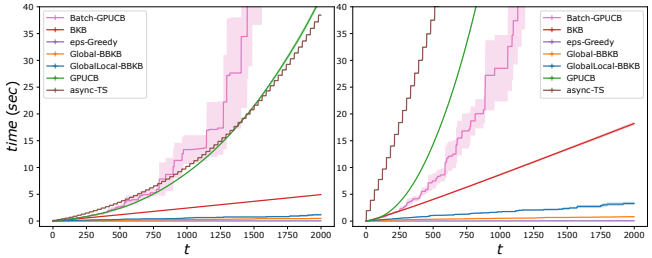

Figure 2: Time on Abalone (left) and Cadata (right)

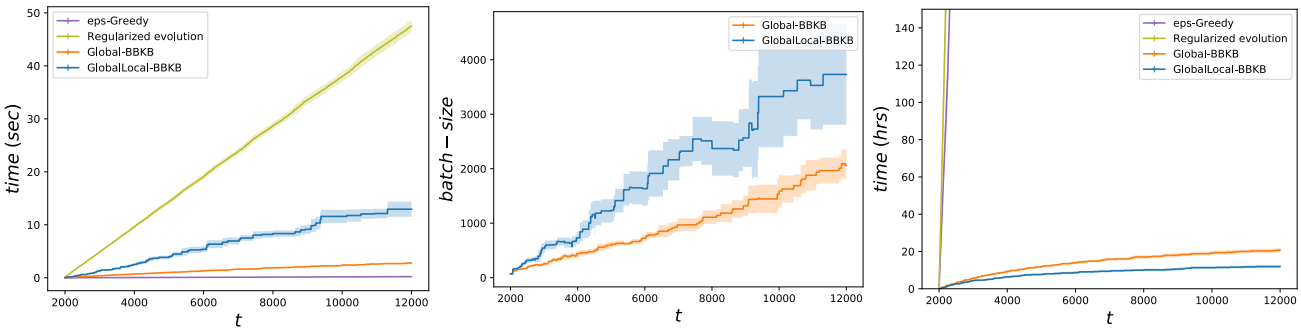

Figure 3: From left to right time without experimental costs, batch-size and total runtime on NAS-bench-101

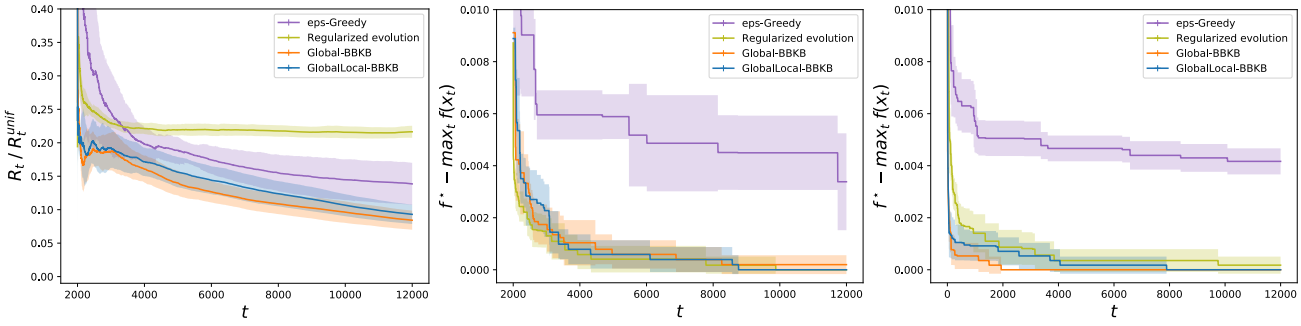

Figure 4: From left to right regret-ratio, simple regret and simple regret without initialization on NAS-bench-101

using parallelism when allowed by the libraries. All algorithm use the hyper-parameters suggested by theory. When not applicable, cross validated parameters that perform the best for each individual algorithm are used (e.g. the kernel bandwidth). All the detailed choices and further experiments are reported in the Appendix D.

We first perform experiments on two regression datasets Abalone ($A = 4177$, $d = 8$) and Cadata ($A = 20640$, $d = 8$) datasets. We first rescale the regression target y to $[0, 1]$, and construct a noisy function to optimize by artificially adding a gaussian noise with zero mean and standard deviation $\xi = 0.01$. For a horizon of $T = 10^4$ iterations, we show in Figure 1 the ratio between the cumulative regret R_t of the desired algorithm and the cumulative regret R_t^{unif} achieved by a baseline policy that selects candidates uniformly at random. We use this metric because it is invariant to the scale of the feedback. We also report in Figure 2 the runtime of the first 2×10^3 iterations. For both datasets, BBKB achieves the smallest regret, using only a fraction of the time of the baselines. Moreover, note that the time reported do not take into account experimentation costs, as the function is evaluated instantly.

To test how much batching can improve experimental runtime, we then perform experiments on the NAS-bench-101 dataset (Ying et al., 2019), a neural network architecture search (NAS) dataset. After preprocessing we are left with $A = 12416$ candidates in $d = 19$ dimensions (details in Appendix D). For each candidate, the dataset contains 3 evaluations of the trained network, which we transform in a noisy function by returning one uniformly at random, and the time required to train the network. To simulate a realistic NAS scenario, we assume to start with already $T_{\text{init}} = 2000$ evaluated network architectures, selected uniformly at random. Initializing BBKB using this data is straightforward.

To generate an initial dictionary we use the BLESS algorithm (Rudi et al., 2018), with a time cost of $2.5s$.
In Figure 3 we first report on the left the runtime of each algorithm without considering experimental costs. While both BBKB variants outperform baselines, due to the more expensive ratio estimator GlobalLocal-BBKB is slower than Global-BBKB. However, while both termination rules guarantee linearly increasing batch-sizes, we can see that the local rule outperforms the global rule. When taking into account training time, this not only shows that the batched algorithm are faster than sequential REG-EVOLUTION, but also that GlobalLocal-BBKB with its larger batches becomes faster than Global-BBKB.

In Figure 4 we report cumulative and simple regret of BBKB against REG-EVOLUTION, the best algorithm from (Ying et al., 2019). To measure the regret, we plot the regret ratio R_t/R_t^{unif} and the simple regret (the gap between the best candidate and the best candidate found by the algorithm up to time t). We consider the simple regret metric because it is used in the NAS-bench-101 paper to evaluate REG-EVOLUTION. From the plot of the regret ratio R_t/R_t^{unif}, we can observe how BBKB is significantly better than REG-EVOLUTION as this latter algorithm has not been designed to minimize the cumulative regret. Further, BBKB is able to match REG-EVOLUTION's simple regret (the main target for this latter algorithm).

Finally, in the rightmost plot of Figure 4 we report simple regret when $T_{\text{init}} = 0$ (i.e., without using initialization). Surprisingly, while the performance of REG-EVOLUTION decreases the performance of BBKB actually increases, outperforming REG-EVOLUTION's. This might hint that initialization is not always beneficial in Bayesian optimization. It remains an open question to verify whether this is because the uniformly sampled initialization data makes the GP harder to approximate, or because it promotes an excessive level of exploration by increasing β_t but reducing variance only in suboptimal parts of $\mathcal{A}$.

6. Conclusions and Open Questions

In this paper we have presented Batched Budgeted Kernelized Bandits (BBKB), a novel efficient GP optimization algorithm with theoretical guarantees. BBKB is currently the most efficient among provably no-regret GP optimization algorithms, with a guaranteed near-linear time complexity.

The key to BBKB's success is based on two new results. First, we have proposed a novel adaptive strategy to select the sizes of the batches of candidates. This strategy makes BBKB more experimentally scalable than existing methods, all while provably preserving selection performance and low regret. Larger batches can then be exploited in the optimization pipeline, and leverage distributed resources to evaluate candidates in parallel resulting in significant wall-clock runtime improvements. Second, we have proved that the same adaptive strategy can be used to delay the updates of BBKB's sparse GP approximation. This choice provides major computational savings without compromising the regret. In the end, we have shown how well BBKB performs in practice, providing a reliable tool that can be used to solve complex and large scale optimization problems such as neural architecture search.

BBKB also opens interesting new opportunities to design a new generation of efficient and theoretically sound GP optimization algorithms. In particular, BBKB was designed with the objective of controlling cumulative regret under frequentist assumptions, i.e., that f has a bounded norm. However, GP-UCB has also been analysed under Bayesian assumptions, were f is sampled from a GP and the objective to control is the expected regret under this prior. This gave rise to Bayesian variants of GP-UCB, which could be also accelerated using the batching and delayed resparsification of BBKB but would require a novel regret analysis.

In this work we have considered only the UCB acquisition function. While it is straightforward to extend our result to Thompson sampling, it remains an important open question to see how we can use other popular Bayesian acquisition functions like expected improvement and knowledge gradient. While these acquisition functions are not known to achieve no-regret in the frequentist setting, some do under different Bayesian assumptions, and can potentially be made scalable using an approach similar to BBKB. Similarly, GP-UCB has been extensively used as a building block in more refined GP optimization heuristics, for example by running multiple local instances of GP-UCB (Wang et al., 2014). Since BBKB has same regret of GP-UCB but is much more efficient, it is natural and immediate to combine it with these heuristics to further improve their scalability.

Finally, note that our theoretical analysis also extends to continuous GP optimization. However, note that the computational bottleneck for the continuous setting is the maximization of the UCBs over a continuous set. Most existing results ignore this aspect and assume that an oracle can provide the maximum in constant time, while in reality it can be quite a complex problem (Kawaguchi et al., 2015; Kirschner et al., 2019). In practice, to maximize the UCBs many software packages exploits sampling techniques or iterative methods such as L-BFGS (Balandat et al., 2019). BBKB's sparse approximation fits very nicely with such approaches, since it provides a finite-dimensional representation of the GP that is amenable to optimization. Moreover, since BBKB's dictionary and sparse representation remain fixed across a large number of iterations, this results in additional benefits such as the ability to warm-start the inner optimization problems.

Acknowledgements

This material is based upon work supported by the Center for Brains, Minds and Machines (CBMM), funded by NSF STC award CCF-1231216, and the Italian Institute of Technology. We gratefully acknowledge the support of NVIDIA Corporation for the donation of the Titan Xp GPUs and the Tesla k40 GPU used for this research. L. R. acknowledges the financial support of the European Research Council (grant SLING 819789), the AFOSR projects FA9550-17-1-0390 and BAA-AFRL-AFOSR-2016-0007 (European Office of Aerospace Research and Development), and the EU H2020-MSCA-RISE project NoMADS - DLV-777826.

References

Abbasi-Yadkori, Y., Pál, D., and Szepesvári, C. Improved algorithms for linear stochastic bandits. In *Advances in Neural Information Processing Systems*, pp. 2312–2320, 2011.

Alaoui, A. E. and Mahoney, M. W. Fast randomized kernel methods with statistical guarantees. In *Neural Information Processing Systems*, 2015.

Balandat, M., Karrer, B., Jiang, D. R., Daulton, S., Letham, B., Wilson, A. G., and Bakshy, E. Botorch: Programmable bayesian optimization in pytorch. *arXiv preprint arXiv:1910.06403*, 2019.

Calandriello, D., Lazaric, A., and Valko, M. Second-order kernel online convex optimization with adaptive sketching. In *International Conference on Machine Learning*, 2017a.

Calandriello, D., Lazaric, A., and Valko, M. Distributed adaptive sampling for kernel matrix approximation. In *AISTATS*, 2017b.

Calandriello, D., Carratino, L., Lazaric, A., Valko, M., and Rosasco, L. Gaussian process optimization with adaptive sketching: Scalable and no regret. In *Conference on Learning Theory*, 2019.

Chevalier, C. and Ginsbourger, D. Fast computation of the multi-points expected improvement with applications in batch selection. In *International Conference on Learning and Intelligent Optimization*, pp. 59–69. Springer, 2013.

Choromanski, K. M., Pacchiano, A., Parker-Holder, J., Tang, Y., and Sindhwani, V. From complexity to simplicity: Adaptive es-active subspaces for blackbox optimization. In *Advances in Neural Information Processing Systems*, pp. 10299–10309, 2019.

Chowdhury, S. R. and Gopalan, A. On kernelized multi-armed bandits. In *International Conference on Machine Learning*, pp. 844–853, 2017.

Contal, E., Buffoni, D., Robicquet, A., and Vayatis, N. Parallel Gaussian process optimization with upper confidence bound and pure exploration. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pp. 225–240. Springer, 2013.

Daxberger, E. A. and Low, B. K. H. Distributed batch Gaussian process optimization. In Precup, D. and Teh, Y. W. (eds.), *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pp. 951–960, International Convention Centre, Sydney, Australia, 06–11 Aug 2017. PMLR.

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.

Hennig, P. and Schuler, C. J. Entropy Search for Information-Efficient Global Optimization. *Journal of Machine Learning Research*, 13:1809–1837, 2012.

Huggins, J. H., Campbell, T., Kasprzak, M., and Broderick, T. Scalable gaussian process inference with finite-data mean and variance guarantees. *International Conference on Artificial Intelligence and Statistics*, 2019.

Jones, D. R., Schonlau, M., and Welch, W. J. Efficient global optimization of expensive black-box functions. *Journal of Global optimization*, 13(4):455–492, 1998.

Kandasamy, K., Krishnamurthy, A., Schneider, J., and Póczos, B. Parallelised bayesian optimisation via thompson sampling. In *International Conference on Artificial Intelligence and Statistics*, pp. 133–142, 2018.

Kathuria, T., Deshpande, A., and Kohli, P. Batched gaussian process bandit optimization via determinantal point processes. In *Advances in Neural Information Processing Systems*, pp. 4206–4214, 2016.

Kawaguchi, K., Kaelbling, L. P., and Lozano-Pérez, T. Bayesian optimization with exponential convergence. In *Advances in neural information processing systems*, pp. 2809–2817, 2015.

Kirschner, J., Mutny, M., Hiller, N., Ischebeck, R., and Krause, A. Adaptive and safe bayesian optimization in high dimensions via one-dimensional subspaces. In *International Conference on Machine Learning*, pp. 3429–3438, 2019.

Lattimore, T. and Szepesvári, C. *Bandit algorithms*. Cambridge University Press, 2020.

Mutny, M. and Krause, A. Efficient High Dimensional Bayesian Optimization with Additivity and Quadrature

Fourier Features. In Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 31*, pp. 9019–9030. Curran Associates, Inc., 2018.

Quinonero-Candela, J., Rasmussen, C. E., and Williams, C. K. Approximation methods for gaussian process regression. *Large-scale kernel machines*, pp. 203–224, 2007.

Rasmussen, C. E. and Williams, C. K. I. *Gaussian processes for machine learning*. Adaptive computation and machine learning. MIT Press, Cambridge, Mass, 2006. ISBN 978-0-262-18253-9. OCLC: ocm61285753.

Real, E., Aggarwal, A., Huang, Y., and Le, Q. V. Regularized evolution for image classifier architecture search. In *Proceedings of the aaai conference on artificial intelligence*, volume 33, pp. 4780–4789, 2019.

Rudi, A., Calandriello, D., Carratino, L., and Rosasco, L. On fast leverage score sampling and optimal learning. In *Advances in Neural Information Processing Systems 31*, pp. 5672–5682. 2018.

Ryzhov, I. O., Powell, W. B., and Frazier, P. I. The knowledge gradient algorithm for a general class of online learning problems. *Operations Research*, 60(1):180–195, 2012.

Seeger, M., Williams, C., and Lawrence, N. Fast forward selection to speed up sparse gaussian process regression. In *Artificial Intelligence and Statistics 9*, number EPFL-CONF-161318, 2003.

Shah, A. and Ghahramani, Z. Parallel predictive entropy search for batch global optimization of expensive objective functions. In *Advances in Neural Information Processing Systems*, pp. 3330–3338, 2015.

Srinivas, N., Krause, A., Seeger, M., and Kakade, S. M. Gaussian process optimization in the bandit setting: No regret and experimental design. In *International Conference on Machine Learning*, pp. 1015–1022, 2010.

Valko, M., Korda, N., Munos, R., Flaounas, I., and Cristianini, N. Finite-time analysis of kernelised contextual bandits. In *Proceedings of the Twenty-Ninth Conference on Uncertainty in Artificial Intelligence*, pp. 654–663. AUAI Press, 2013.

Wang, Z., Shakibi, B., Jin, L., and Freitas, N. Bayesian multi-scale optimistic optimization. In *Artificial Intelligence and Statistics*, pp. 1005–1014, 2014.

Wang, Z., Gehring, C., Kohli, P., and Jegelka, S. Batched large-scale bayesian optimization in high-dimensional spaces. In *International Conference on Artificial Intelligence and Statistics*, pp. 745–754, 2018.

Wilson, A. and Nickisch, H. Kernel interpolation for scalable structured gaussian processes (kiss-gp). In *International Conference on Machine Learning*, pp. 1775–1784, 2015.

Woodruff, D. P. et al. Sketching as a tool for numerical linear algebra. *Foundations and Trends® in Theoretical Computer Science*, 10(1–2):1–157, 2014.

Ying, C., Klein, A., Real, E., Christiansen, E., Murphy, K., and Hutter, F. Nas-bench-101: Towards reproducible neural architecture search. *arXiv preprint arXiv:1902.09635*, 2019.