# Mixed-Variable Bayesian Optimization



**Author(s):**
Daxberger, Erik A.; Makarova, Anastasia; Turchetta, Matteo; Krause, Andreas

**Publication date:**
2019-07-02

**Permanent link:**
https://doi.org/10.3929/ethz-b-000385835

# Mixed-Variable Bayesian Optimization

**Erik Daxberger**[*,†]
University of Cambridge
MPI for Intelligent Systems

**Anastasia Makarova**[*]
ETH Zürich

**Matteo Turchetta**[*]
ETH Zürich

**Andreas Krause**
ETH Zürich

## Abstract

The optimization of expensive to evaluate, black-box, mixed-variable functions, i.e. functions that have continuous and discrete inputs, is a difficult and yet pervasive problem in science and engineering. In Bayesian optimization (BO), special cases of this problem that consider fully continuous or fully discrete domains have been widely studied. However, few methods exist for mixed-variable domains. In this paper, we introduce MiVaBo, a novel BO algorithm for the efficient optimization of mixed-variable functions that combines a linear surrogate model based on expressive feature representations with Thompson sampling. We propose two methods to optimize its acquisition function, a challenging problem for mixed-variable domains, and we show that MiVaBo can handle complex constraints over the discrete part of the domain that other methods cannot take into account. Moreover, we provide the first convergence analysis of a mixed-variable BO algorithm. Finally, we show that MiVaBo is significantly more sample efficient than state-of-the-art mixed-variable BO algorithms on several hyperparameter tuning tasks.

## 1 INTRODUCTION

Bayesian optimization (BO) (Močkus, 1975) is a well-established paradigm to optimize costly-to-evaluate, black-box objectives that has been successfully applied to multiple scientific domains. Most of the existing BO literature focuses on objectives that have purely *continuous* domains, such as those arising in tuning of continuous hyperparameters of machine learning algorithms, recommender systems, and preference learning (Shahriari et al., 2016). More recently, problems with purely *discrete* domains, such as food safety control and model-sparsification in multi-component systems (Baptista and Poloczek, 2018) have been considered.

However, many real-world optimization problems in applied mathematics, engineering and the natural sciences are of *mixed-variable* nature, involving *both* continuous and discrete input variables, and exhibit complex constraints. For example, tuning the hyperparameters of a convolutional neural network involves both continuous variables, e.g., learning rate and momentum, and discrete ones, e.g., kernel size, stride and padding. In addition, these hyperparameters impose validity constrains, e.g., there are combinations of kernel size, stride and padding that lead to invalid networks. Further examples of mixed-variable, potentially constrained, optimization problems include sensor placement (Krause et al., 2008), drug discovery (Negoescu et al., 2011), configuration of optimization solvers (Hutter et al., 2010) and many others. In this work, we introduce an algorithm that can efficiently optimize mixed-variable functions subject to known constraints.

**Related Work.** The efficient optimization of black-box functions over continuous domains has been extensively studied in the BO literature (Srinivas et al., 2010; Wang and Jegelka, 2017; Hennig and Schuler, 2012) as well as the extensions to the problems with unknown constrains (Gardner et al., 2014; Henrández-Lobato et al., 2014; Sui et al., 2015). However, to adapt these methods to the mixed-variable setting, it is necessary to use ad-hoc relaxation techniques to map the problem to a fully continuous one and rounding methods to map the resulting solution to the original domain. This procedure ignores the original structure of the domain and makes the quality of the solution dependent on the choice of relaxation and rounding methods. Moreover, in this setting, it is hard to incorporate known complex constraints over the discrete input variables.

More recently, BO algorithms for discrete domains have been proposed (Baptista and Poloczek, 2018; Oh et al., 2019). However, the application of these methods to the mixed-variable setting requires discretizing the continuous part of the domain, where the discretization granularity

*Equal contribution. †Work done while at ETH Zurich. Correspondence to: Erik Daxberger ead54@cam.ac.uk.

plays a crucial role: If it is too small, it makes the input space prohibitively large; if it is too large, the resulting domain may contain only poorly performing values of the continuous inputs.

Few BO methods can directly solve mixed-variable optimization problems. For example, SMAC (Hutter et al., 2011) uses a random forest as a surrogate model, which accommodates for mixed-variable inputs. However, the frequentist uncertainty estimate provided by random forests may suffer from variance degradation and may not be accurate enough to steer the sampling. The TPE algorithm (Bergstra et al., 2011a) uses non-parametric kernel density estimators (KDEs) to identify inputs that are likely to improve upon and unlikely to perform worse than the best input found so far. Due to the nature of KDEs, TPE naturally supports mixed input spaces. Moreover, its inference step is computationally efficient. While SMAC and TPE can handle hierarchical constraints, they cannot handle more general constraints over the discrete variables. Moreover, they do not have any convergence guarantees. Hyperband (HB) (Li et al., 2016) is a variant of random search that exploits cheap but less accurate approximations of the objective to dynamically allocate resources for function evaluations. BOHB (Falkner et al., 2018) is the model-based counterpart of HB, based on TPE. Therefore, they extend existing mixed-variable optimization methods to the multi-fidelity setting rather than proposing new ones. As such, the core idea of BOHB is complementary to our MIVABO method, rather than in competition with it. Surrogate modeling over a mixed-variable domain is addressed in (Bajer and Holeňa, 2010); however, the authors omit acquisition function optimization, which is a challenging problem, especially in mixed-variable domains. Moreover, the authors leave the important part of choosing the model features to its user. (Garrido-Merchán and Hernández-Lobato, 2018) point out that the posterior GP should be independent on which real input the optimizer suggests as long as it falls in the same rounding interval and fix it with a new kernel. In contrast, we propose to use discrete optimizers for the acquisition function, and thus only make integer evaluations and have no rounding bias. As a result, their method lacks any convergence guarantees and is unable to handle linear and quadratic constraints on the discrete variables.

**Contributions.** We introduce MIVABO, a novel algorithm for the efficient optimization of mixed-variable functions subject to known integer linear and quadratic constraints. It is based on a linear surrogate model that decouples the continuous and discrete components of the function from the mixed one using an expressive feature expansion (Section 3.1). We exploit the ability of this model to efficiently draw samples from the posterior over the objective that can be evaluated at every point in the domain (Section 3.2), by combining it with Thompson sampling. Moreover, we present two alternatives to optimize the resulting acquisition function that can incorporate known linear and quadratic constraints (Section 3.3). To the best of our knowledge, this makes MIVABO the first BO method that can handle constraints over discrete variables. Notice that, while in continuous BO, the optimization of the acquisition function is difficult but has well established solutions, the latter is not the case for the mixed-variable case and solving this problem efficiently is a key challenge. Moreover, we provide the first convergence analysis of a mixed-variable BO algorithm (Section 3.4). Finally, we demonstrate the effectiveness of MIVABO on a wide range of complex hyperparameter tuning tasks, such as deep generative model tuning, where it outperforms state-of-the-art methods and performs comparably to human expert tuning (Section 4).

## 2 PROBLEM STATEMENT

We consider the problem of optimizing an unknown and expensive-to-evaluate, scalar-valued function defined over a mixed-variable domain, accessible through noise-perturbed evaluations and subject to known linear and quadratic constraints. Formally, we aim to solve

$$\min_{\mathbf{x}\in\mathcal{X}} f(\mathbf{x}) \quad \text{s.t. } g^c(\mathbf{x}) \geq 0,\ g^d(\mathbf{x}) \geq 0, \tag{1}$$

where $\mathcal{X} \subseteq \mathcal{X}^c \times \mathcal{X}^d$, with $\mathcal{X}^c$ and $\mathcal{X}^d$ being continuous and discrete subspaces and where $g^c(\mathbf{x}) \geq 0$ and $g^d(\mathbf{x}) \geq 0$ represent a set of known linear and quadratic constraints defined over the continuous and discrete parts of the domain, respectively. We assume that the continuous variables live in a box-constrained domain which, w.l.o.g., can be scaled to the unit hypercube. Therefore, we have $\mathcal{X}^c = [0,1]^{D_c}$. Furthermore, we assume, w.l.o.g., that the discrete variables are binary, i.e., that vectors $\mathbf{x}^d \in \mathcal{X}^d$ correspond to vertices of the unit hypercube. Therefore, we have $\mathcal{X}^d = \{0,1\}^{D_d}$. This binary representation can be used to describe any discrete space, and may thus serve as the domain space of any discrete function. For example, a vector $\mathbf{x}^d = [x_1^d, \ldots, x_{D_d}^d] \in \mathcal{X}^d$ could correspond to a subset $A$ of some ground set $\mathcal{V} = \{v_1, \ldots, v_{D_d}\}$ of $D_d$ discrete elements, such that $x_i^d = 1 \Leftrightarrow a_i \in A$ and $x_i^d = 0 \Leftrightarrow a_i \notin A$, thus yielding a set function. Alternatively, a vector $\mathbf{x}^d \in \mathcal{X}^d$ could correspond to a binary encoding of one or more (non-binary) integer-valued variables, thus resulting in a function defined on an integral domain.

**Background.** Bayesian optimization (BO) algorithms are iterative black-box optimization methods where, at every step $t$, we select an input $\mathbf{x}_t \in \mathcal{X}$ and observe a noise-perturbed output $y_t \triangleq f(\mathbf{x}_t) + \epsilon$ with $\epsilon \overset{\text{iid}}{\sim} \mathcal{N}(0, \beta^{-1})$, where $\beta > 0$. Since evaluating $f$ is costly, the goal is to query inputs based on past observations to find a global minimizer $\mathbf{x}_* \in \arg\min_{\mathbf{x}\in\mathcal{X}} f(\mathbf{x})$ as efficiently and accurately as possible. To this end, BO algorithms leverage

two components: (i) a *probabilistic function model*, also known as surrogate, that encodes the belief about $f$ based on the observations available, and (ii) an *acquisition function* $\alpha : \mathcal{X} \to \mathbb{R}$ that expresses the informativeness of input $\mathbf{x}$ about the location of $\mathbf{x}_*$, given the surrogate of $f$. Based on our belief about the objective, encoded by the probabilistic model, we query the most informative input, measured by the acquisition function. We then update the model with the resulting observation and repeat this procedure. The goal of the acquisition function is to simultaneously learn about the inputs that are likely to be optimal and about poorly explored regions of the input space, i.e., to trade-off exploitation against exploration.

Thus, BO reduces the initial challenging black-box optimization problem to a series of cheaper optimization problems $\mathbf{x}_t \in \arg\max_{\mathbf{x}\in\mathcal{X}} \alpha_t(\mathbf{x})$. However, in our case, these are mixed-variable optimization problems that exhibit linear and quadratic constraints and, therefore, still challenging. In the next section, we present a surrogate model, an acquisition function and two acquisition function optimization routines that, combined, allow us to efficiently solve the problem in Eq. (1).

# 3 MiVaBo ALGORITHM

We first introduce the linear model used to represent the objective (Sec. 3.1) and describe how to do inference for it (Sec. 3.2). We then show how to use Thompson sampling (Thompson, 1933) to suggest informative inputs to query (Sec. 3.3) and, finally, provide a bound on the regret incurred by MiVaBo. (Sec. 3.4).

## 3.1 Model

We introduce a surrogate model of the objective that accounts for both discrete and continuous input variables in a principled way, while balancing the following conflicting goals. On the one hand, we want a complex and expressive model that can capture or, at least, approximate a large class of real-world functions. On the other hand, excessively complex models may render Bayesian inference and constrained optimization of the mixed-variable acquisition computationally infeasible.

Linear models defined over non-linear feature mappings of the inputs, $f(\mathbf{x}) = \mathbf{w}^\top \boldsymbol{\phi}(\mathbf{x})$, are a class of flexible parametric models that strike a good trade-off between model capacity, interpretability and ease of use through the definition of the non-linear features $\boldsymbol{\phi} : \mathcal{X} \to \mathbb{R}^M$. For this class of models, Bayesian inference scales linearly in the number of data points and cubically in the number of features, $M$ (Bishop, 2006). While the complexity of the model is controlled by the number of its features, its capacity depends on their definition. Therefore, to make the design of a set of expressive features more intuitive, we treat separately the contribution to the objective $f$ from the continuous part of the domain, from the discrete one and from the interaction of the two,

$$f(\mathbf{x}) = \sum_{j\in\{d,c,m\}} \mathbf{w}^j \boldsymbol{\phi}^j(\mathbf{x}^j) \tag{2}$$

where, for $j \in \{d, c, m\}$, $\boldsymbol{\phi}^j(\mathbf{x}^j) = [\phi_i^j(\mathbf{x}^j)]_{i=1}^{M_j} \in \mathbb{R}^{M_j}$ and $\mathbf{w}^j \in \mathbb{R}^{M_j}$ denote the feature and weight vector for the discrete, continuous and mixed component, respectively. A crucial advantage of this model is that, given a sample from the posterior distribution over the weights, the corresponding function can be easily evaluated at any point in the domain. As we will see in Section 3.3, this plays a fundamental role for the optimization of the acquisition function.

In many real-world problems, a large portion of possible features can be discarded a priori, simplifying the design space. A common assumption in BO (especially in high-dimensional BO) is that, for real-world functions, only low-order interactions between the variables contribute significantly to the objective function. This was shown to be the case for many practical problems (Hoang et al., 2018; Rolland et al., 2018; Mutný and Krause, 2018), including hyperparameter tuning of deep neural networks (Hazan et al., 2017). Based on this assumption, we focus on features defined over small subsets of the input variables. Formally, we consider $\boldsymbol{\phi}(\mathbf{x}) = [\phi_k(\mathbf{x}_k)]_{k=1}^M$, where $\mathbf{x}_k$ is a subvector of $\mathbf{x}$ which may contain exclusively continuous or discrete variables or a mix of both. Accordingly, the objective $f(\mathbf{x})$ can be decomposed into a sum of functions $f_k(\mathbf{x}_k) \triangleq w_k \phi_k(\mathbf{x}_k)$ defined over subvectors $\mathbf{x}_k$ from low-dimensional subspaces $\mathcal{X}_k \subseteq \mathcal{X}$ with $\dim(\mathcal{X}_k) \ll \dim(\mathcal{X})$. Such a model decomposition is also known as a *generalized additive model* (Rolland et al., 2018; Hastie, 2017), where it is assumed that a variable can be included in more than one subvector, i.e. $j \neq k \implies \mathcal{X}_j \cap \mathcal{X}_k = \emptyset$. The complexity of such a model can be controlled by the effective dimensionality of the subspaces. This is important in case of computational resource restrictions. In particular, let $\bar{D}_d \triangleq \max_{k\in[M]} \dim(\mathcal{X}_k^d)$ denote the effective dimensionality of the discrete component in Eq. (2), i.e. the dimensionality of the largest among all subspaces that exclusively contains discrete variables. Analogously, $\bar{D}_c$ and $\bar{D}_m$ denote the effective continuous and mixed dimensionalities, respectively. Intuitively, the effective dimensionality corresponds to the maximum order of the variable interactions present in $f$. The number of features $M = \mathcal{O}\big(D_d^{\bar{D}_d} + D_c^{\bar{D}_c} + (D_d + D_c)^{\bar{D}_m}\big)$ then scales exponentially in the effective dimensionalities only[1], which are usually small, even if the true dimensionality is large.

**Discrete Features $\boldsymbol{\phi}^d$.** We aim at defining a set of features, $\phi^d$, that can effectively represent the discrete component of Eq. (2) as a linear function. Generally, the model

[1] I.e., since modelling up to $L$-th order interactions of $N$ variables requires $\sum_{l=0}^{L} \binom{N}{l} \in \mathcal{O}(N^L)$ terms.

should be able to capture arbitrary interactions between the discrete variables. To this end, we consider all subsets $S$ of the discrete variables in $\mathcal{X}^d$ (or equivalently, all elements $S$ of the powerset $2^{\mathcal{X}_d}$ of $\mathcal{X}_d$) and define a monomial $\prod_{j\in S} x_j^d$ for each subset $S$ (where for the empty set, $\prod_{j\in\emptyset} x_j^d = 1$). We then construct a weighted sum of all these monomials to arrive at the multi-linear polynomial $\mathbf{w}^{d\top}\boldsymbol{\phi}^d(\mathbf{x}^d) = \sum_{S\in 2^{\mathcal{X}_d}} w_S \prod_{j\in S} x_j^d$. This functional representation also corresponds to the Fourier expansion of a so-called *pseudo-Boolean function* (PBF) (Boros and Hammer, 2002; O'Donnell, 2014). In practice, an exponential number of features can be prohibitively expensive and may lead to high-variance estimators as in BO one typically does not have access to massive amounts of data to robustly fit a large model (Gelman et al., 1995). Alternatively, (Baptista and Poloczek, 2018; Hazan et al., 2017) empirically found that a second-order polynomial in the Fourier basis provides a practical balance between expressiveness and efficiency, even when the true function is of higher order. In our model, we also consider quadratic PBFs, $\mathbf{w}^{d\top}\boldsymbol{\phi}^d(\mathbf{x}^d) = w_\emptyset + \sum_{i=1}^n w_{\{i\}} x_i^d + \sum_{1\le i<j\le n} w_{\{i,j\}} x_i^d x_j^d$, which induces the discrete feature representation $\boldsymbol{\phi}^d(\mathbf{x}^d) \triangleq [1, \{x_i^d\}_{i=1}^{D_d}, \{x_i^d x_j^d\}_{1\le i<j\le D_d}]^\top$ and reduces the number of weights in the surrogate model to $M_d \in \mathcal{O}(D_d^2)$.

**Continuous Features $\boldsymbol{\phi}^c$.** In BO over continuous spaces, most approaches are based on non-parametric Gaussian process (GP) models (Williams and Rasmussen, 2006) due to their flexibility and ability to capture large classes of continuous functions. To fit our linear model formulation, we leverage GPs' expressiveness by modeling the continuous part of our model in Eq. (2) using feature expansions $\boldsymbol{\phi}^c(\mathbf{x}^c)$ that result in a finite linear approximation of a GP. One conceptually simple yet theoretically sound choice of such a basis expansion, which we also use in our experiments, is the celebrated class of Random Fourier Features (RFF) (Rahimi and Recht, 2008; Rahimi and Recht, 2009), which induce a randomized approximation of a GP based on Monte Carlo integration. Alternatively, one can leverage Quadrature Fourier Features (QFF) (Mutný and Krause, 2018), which instead use a deterministic approximation based on numerical integration, and which are particularly effective for problems with low effective dimensionality. Both methods have been successfully applied in BO (Jenatton et al., 2017; Perrone et al., 2017; Mutný and Krause, 2018). Alternatively, one can also use features learned from data via a neural network (Snoek et al., 2015).

**Mixed Features $\boldsymbol{\phi}^m$.** The goal of the mixed term is to capture as rich and realistic interactions between the discrete and continuous variables as possible, while keeping model inference and acquisition function optimization efficient. To this end, we stack products of all pairwise combinations of features of the two variable types, i.e. $\boldsymbol{\phi}^m(\mathbf{x}^d, \mathbf{x}^c) \triangleq [\phi_i^d(\mathbf{x}^d)\cdot\phi_j^c(\mathbf{x}^c)]_{1\le i\le M_d, 1\le j\le M_c}^\top$. This formulation provides a good trade-off between modeling accuracy and computational complexity. In particular, is allows us to reduce $\boldsymbol{\phi}^m$ to the discrete feature representation $\boldsymbol{\phi}^d$ when conditioned on a fixed assignment of continuous variables $\boldsymbol{\phi}^c$ (and vice versa). This property is of central importance for optimizing the acquisition function, as it allows us to optimize the mixed term of our model by leveraging the tools available for optimizing the discrete and continuous parts individually.

The proposed representation contains $M_m = M_d M_c$ features, thus resulting in a total of $M = M_d + M_c + M_d M_c$. To reduce model complexity, prior knowledge about the problem domain can be incorporated into the construction of the mixed features. In particular, one may consider the following approaches. Firstly, one can exploit a known interaction structure between variables, e.g., in form of a dependency graph, and ignore the features that are known to be irrelevant. Secondly, one can start by including all of the proposed pairwise feature combinations and progressively discard not-promising ones. Finally, for high-dimensional problems, one can use the opposite strategy and progressively add pairwise feature combinations, starting from the empty set.

### 3.2 Model Inference

**Probabilistic Model Formulation.** Let $\mathbf{X}_{1:t} \in \mathbb{R}^{t\times D}$ denote the matrix whose $i^{th}$ row contains the input $\mathbf{x}_i \in \mathcal{X}$ queried at iteration $i$, $\dim\mathcal{X} = D$, and let $\mathbf{y}_{1:t} = [y_1, \dots, y_t]^\top \in \mathbb{R}^t$ denote the array containing the corresponding noisy function observations. Furthermore, let $\boldsymbol{\Phi}_{1:t} \in \mathbb{R}^{t\times M}$ denote the matrix whose $i^{th}$ row contains the featurized input $\boldsymbol{\phi}(\mathbf{x}_i) \in \mathbb{R}^M$. Given the proposed formulation of $f$ in Eq. (2) together with the noisy observation model, we obtain the Gaussian likelihood $p(\mathbf{y}_{1:t}|\mathbf{X}_{1:t}, \mathbf{w}) = \mathcal{N}(\boldsymbol{\Phi}_{1:t}\mathbf{w}, \beta^{-1}\mathbf{I})$. To complete our model, we specify a prior distribution $p(\mathbf{w}|\alpha)$ over the coefficient vector $\mathbf{w}$ parametrised by some scale parameter $\alpha > 0$. This prior should reflect our *a priori* knowledge about $\mathbf{w}$ and thus the objective $f$. Given the likelihood and prior, the goal is to infer the posterior distribution $p(\mathbf{w}|\mathbf{X}_{1:t}, \mathbf{y}_{1:t}, \alpha, \beta) \propto p(\mathbf{y}_{1:t}|\mathbf{X}_{1:t}, \mathbf{w}, \beta)p(\mathbf{w}|\alpha)$. The difficulty of this crucially depends on the choice of prior, for which we present two viable alternatives.

**Gaussian Prior.** One natural way to complete a Gaussian likelihood model is to employ a zero-mean isotropic Gaussian prior distribution on the weight vector $\mathbf{w}$, i.e., $p(\mathbf{w}|\alpha) = \mathcal{N}(\mathbf{0}, \alpha^{-1}\mathbf{I})$, with precision $\alpha > 0$. A Gaussian prior encourages the weights to be uniformly small in size, so that the final predictor is a sum of many (or all) features, with each giving a small but non-zero contribution. Due to conjugacy, a Bayesian treatment of the weights $\mathbf{w}$ yields a Gaussian posterior distribution $p(\mathbf{w}|\mathbf{X}_{1:t}, \mathbf{y}_{1:t}) = \mathcal{N}(\mathbf{m}, \mathbf{S}^{-1})$ with mean $\mathbf{m} = \beta\mathbf{S}^{-1}\boldsymbol{\Phi}_{1:t}^\top\mathbf{y}_{1:t} \in \mathbb{R}^M$ and in-

verse covariance $\mathbf{S} = \alpha\mathbf{I} + \beta\mathbf{\Phi}_{1:t}^\top\mathbf{\Phi}_{1:t} \in \mathbb{R}^{M\times M}$ (Bishop, 2006). This simple analytical treatment of the posterior is a main benefit of this model, which can be viewed as a GP with a linear kernel in feature space.

**Sparse Prior.** While the number and degree of the features used in the model is a design choice, in practice it is typically unknown which variable interactions matter and thus which features to choose. To discard irrelevant features, one may impose a sparsity-encouraging prior over the weight vector $\mathbf{w}$ (Baptista and Poloczek, 2018). However, due to non-conjugacy to the Gaussian likelihood, exact Bayesian inference of the resulting posterior distribution is in general intractable, imposing the need for approximate inference methods. One choice for such a prior is the Laplace distribution, for which approximate inference techniques based on expectation propagation (Minka, 2001) and variational inference (Wainwright et al., 2008) were developed in (Seeger, 2008; Seeger and Nickisch, 2008; Seeger and Nickisch, 2011). Alternatively, one can use a horseshoe prior and use Gibbs sampling to sample from the posterior over weights (Baptista and Poloczek, 2018). However, this comes with a significantly larger computational burden, which is a well-known issue for sampling based inference techniques (Bishop, 2006). Lastly, one may consider a spike-and-slab prior with expectation propagation for approximate posterior inference (Hernández-Lobato et al., 2013; Hernández-Lobato et al., 2015).

### 3.3 Acquisition Function

We propose to use Thompson sampling (TS) (Thompson, 1933), a technique which samples weights $\widetilde{\mathbf{w}} \sim p(\mathbf{w}|\mathbf{X}_{1:t}, \mathbf{y}_{1:t}, \alpha, \beta)$ from the posterior distribution and chooses the next input by solving $\hat{\mathbf{x}} \in \arg\max_{\mathbf{x}\in\mathcal{X}} \widetilde{\mathbf{w}}^\top\boldsymbol{\phi}(\mathbf{x})$ (see Algorithm 1 in Appendix C). Intuitively, by sampling from the posterior, TS focuses on inputs that are plausibly optimal.

TS has previously been successfully applied in both discrete and continuous domains (Baptista and Poloczek, 2018; Mutný and Krause, 2018). In addition, the following considerations make TS an ideal acquisition function for our problem. Firstly, the simple relation between the surrogate model and the resulting optimization problem for the acquisition function allows us to trade off model expressiveness and optimization tractability, which is a key challenge in mixed-variable domains. In particular, if we model second-order discrete interactions, the optimization of the acquisition function requires us to solve a quadratic mixed integer program, which can be done using available solvers and which allows us to incorporate complex constraints on the discrete variables. Secondly, in combination with a linear surrogate model it allows us to provide a convergence analysis, making MiVaBo the first mixed-variable BO method that enjoys theoretical guarantees.

Our acquisition strategy requires solving $\hat{\mathbf{x}} \in \arg\max_{\mathbf{x}\in\mathcal{X}} \tilde{\mathbf{w}}_t^\top\boldsymbol{\phi}(\mathbf{x})$, which is a challenging mixed-variable optimization problem. To this end, we propose two schemes – alternating optimization and dual decomposition – which leverage independent subroutines for discrete and continuous optimization. While alternating optimization is a simple and efficient method that often works well in practice, dual decomposition comes with theoretical guarantees but is more expensive to run. Thus, in practice, one can choose the method to use based on the requirements of application at hand.

For the discrete part, we exploit of the fact that the optimization of a second-order pseudo-Boolean function can be formulated as a binary integer quadratic program (IQP) (Boros and Hammer, 2002), allowing us to exploit commonly-used efficient and robust optimization tools, such as Gurobi (Optimization, 2014) or CPLEX (IBM, 2009).[2] This approach allows us to use any functionality offered by these solvers, such as the ability to optimize objectives subject to linear constraints $\mathbf{A}\mathbf{x}^d \leq \mathbf{b}$, $\mathbf{A} \in \mathbb{R}^{K\times D_d}, \mathbf{b} \in \mathbb{R}^K$ or quadratic constraints ${\mathbf{x}^d}^\top\mathbf{Q}\mathbf{x}^d + \mathbf{q}^\top\mathbf{x}^d \leq b$, $\mathbf{Q} \in \mathbb{R}^{D_d\times D_d}, \mathbf{q} \in \mathbb{R}^{D_d}, b \in \mathbb{R}$. For the continuous part, one can use optimizers commonly used in continuous BO, such as L-BFGS (Liu and Nocedal, 1989) or DIRECT (Jones, 2001).

**Alternating Optimization**. One natural way to optimize an objective in Eq. (2) is an alternating optimization scheme which iterates between optimizing the discrete variables $\mathbf{x}^d$ conditioned on a particular setting of the continuous variables $\mathbf{x}^c$ and vice versa, until convergence to some local optimum. While these approaches often provide no theoretical guarantees, they are widely applied in many contexts where the objective function is hard to optimize. In particular, we iteratively solve $\hat{\mathbf{x}}^d \in \arg\max_{\mathbf{x}^d\in\mathcal{X}^d} \left({\tilde{\mathbf{w}}^d}^\top\boldsymbol{\phi}^d(\mathbf{x}^d) + {\tilde{\mathbf{w}}^m}^\top\boldsymbol{\phi}^m(\mathbf{x}^d, \mathbf{x}^c = \tilde{\mathbf{x}}^c)\right)$ on the discrete domain and $\hat{\mathbf{x}}^c \in \arg\max_{\mathbf{x}^c\in\mathcal{X}^c} \left({\tilde{\mathbf{w}}^c}^\top\boldsymbol{\phi}^c(\mathbf{x}^c) + {\tilde{\mathbf{w}}^m}^\top\boldsymbol{\phi}^m(\mathbf{x}^d = \tilde{\mathbf{x}}^d, \mathbf{x}^c)\right)$ on the continuous domain. Importantly, the mixed features from Section 3.1 reduce these maximization problems to those that can be optimized by the oracles for discrete and continuous optimization.

**Optimization with theoretical guarantees.** Alternatively, one can maximize Eq. (2) using dual decomposition, which is a powerful approach based on Lagrangian optimization and has been successfully used for many problems (Komodakis et al., 2011; Sontag et al., 2011; Rush and Collins, 2012). Its well-studied theoretical properties facilitate the convergence analysis of MiVaBo, making it particularly useful for settings where optimization accuracy is of cru-

[2] While solving general binary IQPs is well-known to be NP-hard (Boros and Hammer, 2002), these solvers are in practice very efficient for the problem dimensionalities we consider (i.e., with $D_d < 100$).

cial importance. Due to lack of space, we refer the reader to Appendix D for details on the dual decomposition and its derivation for our problem.

### 3.4 Convergence Analysis

The choice of a linear model and Thompson sampling allows us to leverage convergence analysis from linearly parameterized multi-armed bandits, which are a well-studied class of methods for solving structured decision making problems (Abeille et al., 2017; Agrawal and Goyal, 2013). As in our setting, these methods assume the objective to be a linear function of features $\boldsymbol{\phi}(\mathbf{x}) \in \mathbb{R}^M$ with a fixed but unknown parameter vector $\mathbf{w} \in \mathbb{R}^M$, i.e. $\mathbb{E}[f(\mathbf{x})|\boldsymbol{\phi}(\mathbf{x})] = \mathbf{w}^\top \boldsymbol{\phi}(\mathbf{x})$, and aim at minimizing the *total regret* up to time $T$: $\mathcal{R}(T) = \sum_{t=1}^{T}(f(\mathbf{x}_*) - f(\mathbf{x}_t))$. We obtain the following probabilistic regret bound for MIVABO:

**Proposition 1** *Assume that the following assumptions hold in every iteration $t = 1, \dots, T$:*

1. $\|\mathbf{w}_t\|_2 \leq c, \|\boldsymbol{\phi}(\mathbf{x}_t)\|_2 \leq c, \|f(\mathbf{x}_*) - f(\mathbf{x}_t)\|_2 \leq c$ , *for some $c \in \mathbb{R}^+$.*
2. $\mathbf{x}_t = \arg\max_{\mathbf{x}} \tilde{\mathbf{w}}^\top \boldsymbol{\phi}(\mathbf{x})$ *is selected exactly.*[3]
3. $\tilde{\mathbf{w}}_t \sim \mathcal{N}(\mathbf{m}, 24M \ln T \ln \frac{1}{\delta}\mathbf{S}^{-1})$*, i.e., the posterior variance $\mathbf{S}^{-1}$ is scaled accordingly.*

*Then, the total regret of* MIVABO *is bounded by $\mathcal{R}(T) \leq \tilde{\mathcal{O}}\left(M^{3/2}\sqrt{T}\ln\frac{1}{\delta}\right)$ with probability $1-\delta$.*

Proposition 1 follows directly from Theorem 1 in (Abeille et al., 2017) is applicable to an infinite number of arms/inputs, i.e. where $\mathbf{x} \in \mathcal{X}, |\mathcal{X}| = \infty$. In our setting, each arm/input corresponds to its feature vector and, as the feature space satisfies the compactness assumption, the proof indeed holds.

Proposition 1 implies *no-regret*, $\lim_{T\to\infty} \mathcal{R}(T)/T = 0$, i.e., convergence to the global maximum, since the maximum found after $T$ iterations is no further away from $f(\mathbf{x}_*)$ than the average regret $\mathcal{R}(T)/T$. To the best of our knowledge, MIVABO is the first mixed-variable BO algorithm for which such a guarantee is known to hold.

## 4 EXPERIMENTS

In this section, we present experimental results on tuning the hyperparameters of two machine learning algorithms, namely gradient boosting and a deep generative model, on multiple data sets. Further empirical results on synthetic benchmarks can be found in Appendix A.

[3]Note that while this is not generally true for alternating optimization, it is guaranteed for dual decomposition under certain conditions.

**MIVABO.** For the continuous model part, we employ Random Fourier Features (RFFs) approximating a GP with a squared exponential (SE) kernel, as we found RFFs to provide the best trade-off between complexity and accuracy in practice. We use the alternating optimization scheme for Thompson sampling, which we always run until convergence to a local optimum. We use Gurobi (Optimization, 2014) as the discrete optimization oracle, and L-BFGS (Liu and Nocedal, 1989) as the continuous optimization oracle. We did not tune any parameters, but left the prior variance $\alpha$, observation noise variance $\beta$, and kernel bandwidth $\sigma$ at 1.0, and scaled the posterior variance by the factor stated in Proposition 1. We impose a Gaussian prior over the weights, allowing us to perform closed-form inference. We also ran experiments with the Laplace sparsity-inducing prior. However, while it often yielded better results, this marginal performance gain came with the significantly increased computational effort required to approximate the posterior.

**Baselines.** We compare against four baselines: SMAC (Hutter et al., 2011), TPE (Bergstra et al., 2011b), random search (Bergstra and Bengio, 2012), and the popular GPyOpt open-source BO `Python` package (González, 2016). GPyOpt is based on a Gaussian process (GP) model, which constitutes the most commonly used approach in practice. To account for mixed variable types, GPyOpt relaxes discrete variables to be continuous and later rounds them to the nearest discrete neighbor. In order to separate the influence of the model choice and acquisition function optimization, we also consider the MIVABO model optimized by simulated annealing (SA) (Kirkpatrick et al., 1983) (denoted by MIVABO-SA) as well as a GP model with the upper confidence bound (UCB) acquisition function (Srinivas et al., 2010) optimized by SA (denoted by GP-SA). To handle constraints, SA assigns high energy values to invalid inputs, making the probability of moving there acquisition function optimizer negligible. We use SMAC, TPE and GPyOpt and SA with their respective default settings.[4] We compare against GP-SA and MIVABO-SA only in constrained settings, using more principled methods in unconstrained ones.

**(a) Results for gradient boosting hyperparameter tuning.** We use the publicly available OpenML database (Vanschoren et al., 2014), which contains evaluations for various machine learning methods trained on several datasets with many hyperparameter settings. We consider one of the most popular algorithms from OpenML (in terms of number of evaluations), namely XGBoost, which is an efficient implementation of the extreme gradient boosting framework from (Chen and Guestrin, 2016). The task is to tune

[4]See Appendix F for details on the implementations we used. Since we do not consider settings where cheap approximations of the objective are available, we cannot compare against HB or BOHB.

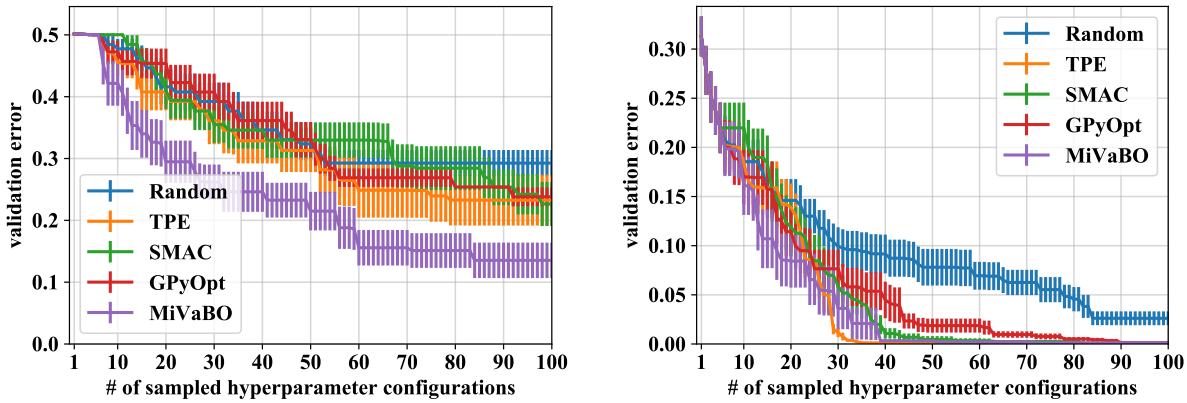


Figure 1: **XGBoost hyperparameter tuning**. Results on the `monks-problem-1` (left) and `steel-plates-fault` (right) datasets. Mean plus/minus one standard deviation of the validation error over 16 random initializations. MiVaBo significantly outperforms the competing state-of-the-art methods on the first dataset, while being competitive on the second.

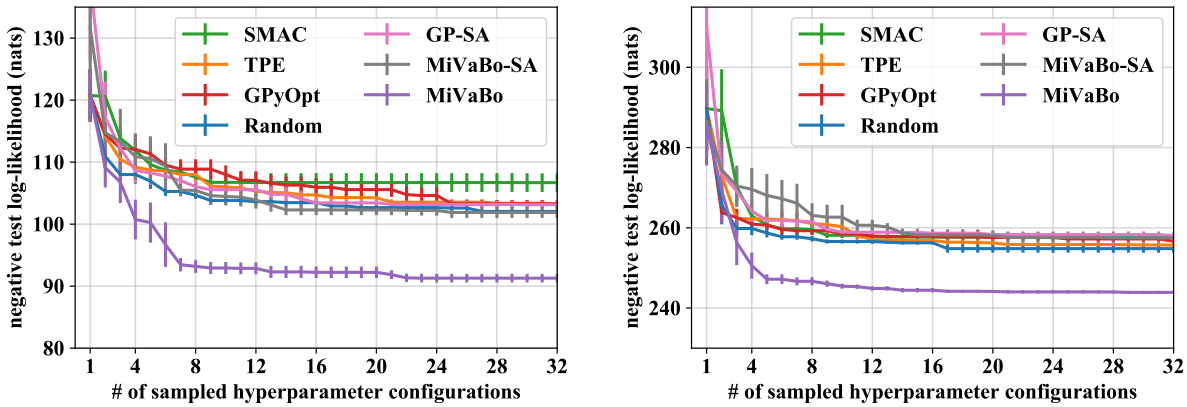


Figure 2: **VAE hyperparameter tuning.** Results on the `MNIST` (left) and `FashionMNIST` (right) datasets. Mean plus/minus one standard deviation of the negative test log-likelihood (NLL) in nats, estimated using 32 importance samples, over 8 random initializations. Every model was trained for 32 epochs. MiVaBo significantly outperforms the competing state-of-the-art methods, demonstrating its ability to handle the complex constrained nature of the VAE's parameter space.

its ten hyperparameters - three are discrete and seven continuous - to minimize the classification error on a held-out test set. We consider two different datasets, each containing more than 45000 hyperparameter evaluations. See Appendix F for more details on XGBoost and the datasets used. To evaluate settings for which no data is available, we use a surrogate modeling approach (Eggensperger et al., 2015) based on nearest neighbor; i.e., for a given hyperparameter setting, the objective returns the error of the closest (in terms of Euclidean distance) setting available in the database. The results in Fig. 1 show that MiVaBo achieves performance which is either significantly stronger than (left plot) or competitive with (right plot) the state-of-the-art mixed-variable BO algorithms on this challenging task (depending on the dataset used for evaluation). GPyOpt performs poorly, which is likely due to the fact that it cannot account for the discrete variables in a principled way. As compared to TPE and SMAC, our method likely benefits from more sophisticated uncertainty estimation.

**(b) Results for deep generative model hyperparameter tuning.** Deep generative models have recently received considerable attention in the machine learning community. Despite their popularity and importance, effectively tuning their hyperparameters is a major challenge. We consider the task of tuning the hyperparameters of a deep neural network based variational auto-encoder (VAE) (Kingma and Welling, 2013) composed of a convolutional encoder and a deconvolutional decoder, as considered in (Dosovitskiy et al., 2015; Salimans et al., 2014). The VAEs are evaluated on the standard train/test split of the stochastically binarized version of the `MNIST` dataset (LeCun et al., 2010), as considered in (Burda et al., 2015; Rainforth et al., 2018), as well as the `FashionMNIST` dataset (Xiao et al., 2017). The models are trained on 60000 images for 32 epochs, using Adam (Kingma and Ba, 2014) with a mini-batch size

Table 1: For the best models found by each algorithm in Fig. 2 (left) after 32 BO iterations, we report the negative log-likelihood (NLL), estimated with 5000 importance samples, after training for 3280 epochs. We also report the NLL achieved after 32 training epochs, as in Fig. 2 (left)

| Algorithm | NLL (nats) | |
|---|---|---|
| | 32 epochs | 3280 ep. |
| **SMAC** | $106.69 \pm 1.51$ | 99.09 |
| **TPE** | $103.28 \pm 0.49$ | 97.05 |
| **GPyOpt** | $103.25 \pm 0.58$ | 97.33 |
| **Random** | $102.04 \pm 0.41$ | 93.74 |
| **MIVABO** | $\mathbf{91.29 \pm 0.69}$ | **84.25** |

Figure 3: Randomly chosen `MNIST` test images (left column) and their reconstructions by the best VAE models found by MIVABO, random search, GPyOpt, TPE and SMAC (from left to right), thus ordered by NLL values, which seem to capture quality of visual appearance.

of 128. We report the negative log-likelihood (in nats) achieved by the VAEs on a held-out test set of 10000 images, as estimated via importance sampling (Rezende et al., 2014) using 32 importance samples per test point. To the best of our knowledge, this is the first BO paper that considers tuning a deep generative model.

The tuning of this VAE is difficult due to the high-dimensional and structured nature of its hyperparameter space, and, in particular, due to the constraints arising from mutual dependencies between some of its parameters. We tune 25 discrete parameters defining the model architecture, including the number of convolutional layers, their stride, padding and filter size, the number of fully-connected layers and their number of neurons, as well as the dimensionality of the VAE's latent space. We furthermore tune three continuous parameters controlling the optimizer and regularization. Crucially, the discrete parameters exhibit mutual dependencies which result in complex constraints in hyperparameter space, as certain combinations of stride, padding and filter size lead to invalid architectures. Particularly, for the encoder, the shapes of all layers must be integral, and for the decoder, the output shape should match the shape of the input data, i.e., one channel of size $28 \times 28$ for `MNIST` and `FashionMNIST`. The latter constraint is especially challenging, as only a small number of possible decoder configurations yield the required output shape. See Appendix B for more details on this experiment.

While MIVABO can conveniently capture these restrictions via linear and quadratic constraints, the competing methods cannot, which makes a fair comparison difficult. We thus adapt the other methods to handle constraints as follows: If a method tries to evaluate an invalid parameter configuration, we return a penalty error value, which will discourage a model-based method to sample this (or a similar) setting again. However, for fairness, we only report valid observations and ignore all configurations that violated a constraint. We set the penalty value to the error incurred by a uniformly random generator, i.e., 500 nats. In Appendix B.3, we investigate the impact of the penalty value on the quality of the solution obtained by these methods and we show that it does not qualitatively affect the results.

The results are presented in Fig. 2 for the `MNIST` (left) and `FashionMNIST` (right) datasets. It can be seen that MIVABO significantly outperforms the competing methods on this task, on both datasets. This is because MIVABO is able to naturally encode the constraints and thus to directly optimize over the feasible region in parameter space, while TPE, SMAC and GPyOpt need to learn the constraints from data. They fail to do so and get stuck in bad local optima early on. The model-based approaches likely have difficulties due to sharp discontinuities in hyperparameter space induced by the constraint violation penalties (i.e., as invalid configurations may lie close to well-performing configurations). In contrast, random search is agnostic to these discontinuities, and thus notably outperforms the model-based methods. Lastly, both baselines employing simulated annealing (i.e., GP-SA and MIVABO-SA) struggle on this task, which suggests that while simulated annealing can avoid invalid inputs, the effective optimization of objectives involving complex constraints crucially requires more principled approaches for acquisition function optimization, such as the ones we propose.

Table 1 shows that the best VAE found by MIVABO achieves $84.25$ nats on `MNIST` when trained for 3280 epochs and using 5000 importance samples for test log-likelihood estimation (which is the setting used in (Burda et al., 2015)). This is comparable to the performance achieved by models tuned by human experts, as, e.g., reported in (Salimans et al., 2014) (where three convolutional layers and a more sophisticated inference procedure are used). This highlights the effectiveness of MIVABO in tuning deep generative models parameterized by convolutional architectures, especially considering the significantly worse performance of the baselines.[5] While log-likelihood

[5]Table 1 also shows that the relative performance differences between the competing methods after only 32 training epochs

scores allow for a quantitative comparison, they are hard to interpret for humans. Thus, for a qualitative comparison, Fig. 3 visualizes the reconstruction quality achieved on `MNIST` by the best VAE configuration found by the different methods after 32 BO iterations. The VAEs were trained for 32 epochs each, as in the BO experiment. The log-likelihoods seem to correlate with the quality of visual appearance, and the model found by MiVaBo produces the visually most appealing reconstructions among all models. Appendix B shows additional plots.

## 5 CONCLUSION

We propose MiVaBo, a simple yet effective method for efficient optimization of expensive-to-evaluate mixed-variable black-box objective functions that can handle linear and quadratic constraints over the discrete part of the domain, combining a linear model of expressive features with Thompson sampling. Our method is highly flexible due to the modularity of its components, i.e., the feature mapping used to model the mixed-input objective, and the optimization oracles used as subroutines for the acquisition procedure. This allows practitioners to tailor MiVaBo to specific objectives, e.g. by incorporating prior knowledge in the feature design or by exploiting optimization oracles that can handle specific types of constraints. Moreover, we show that MiVaBo enjoys strong theoretical convergence guarantees that competing methods lack. Finally, we empirically demonstrate that MiVaBo significantly improves optimization performance as compared to state-of-the-art data driven methods for mixed-variable optimization on complex hyperparameter tuning tasks.

**Acknowledgements**

This research has been partially supported by SNSF NFP75 grant 407540_167189. Matteo Turchetta was supported through the ETH-MPI Center for Learning Systems. The authors would like to thank Josip Djolonga, Mojmír Mutný, Johannes Kirschner, Alonso Marco Valle, David R. Burt, Ross Clarke, Wolfgang Roth as well as the anonymous reviewers of an earlier version of this paper for their helpful feedback.

## References

Abeille, M., Lazaric, A., et al. (2017). Linear Thompson sampling revisited. *Electronic Journal of Statistics*.

Agrawal, S. and Goyal, N. (2013). Thompson sampling for contextual bandits with linear payoffs. In *International conference on Machine learning (ICML)*.

provide a reasonable proxy for the performance differences after 3280 training epochs. This observation significantly speeds up the model tuning process by around $100\times$.

Bajer, L. and Holeňa, M. (2010). Surrogate model for continuous and discrete genetic optimization based on rbf networks. In *Proceedings of the 11th International Conference on Intelligent Data Engineering and Automated Learning*.

Baptista, R. and Poloczek, M. (2018). Bayesian optimization of combinatorial structures. *arXiv preprint arXiv:1806.08838*.

Bergstra, J., Bardenet, R., Kégl, B., and Bengio, Y. (2011a). Implementations of algorithms for hyper-parameter optimization. In *NIPS Workshop on Bayesian optimization*.

Bergstra, J. and Bengio, Y. (2012). Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(Feb):281–305.

Bergstra, J. S., Bardenet, R., Bengio, Y., and Kégl, B. (2011b). Algorithms for hyper-parameter optimization. In *Advances in Neural Information Processing Systems (NIPS)*.

Bishop, C. M. (2006). *Pattern recognition and machine learning*. Springer.

Boros, E. and Hammer, P. L. (2002). Pseudo-boolean optimization. *Discrete applied mathematics*, 123(1-3):155–225.

Burda, Y., Grosse, R., and Salakhutdinov, R. (2015). Importance weighted autoencoders. *arXiv preprint arXiv:1509.00519*.

Chen, T. and Guestrin, C. (2016). Xgboost: A scalable tree boosting system. In *International Conference on Knowledge Discovery and Data Mining*.

Dosovitskiy, A., Tobias Springenberg, J., and Brox, T. (2015). Learning to generate chairs with convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1538–1546.

Eggensperger, K., Hutter, F., Hoos, H. H., and Leyton-Brown, K. (2015). Efficient benchmarking of hyperparameter optimizers via surrogates. In *AAAI*, pages 1114–1120.

Falkner, S., Klein, A., and Hutter, F. (2018). BOHB: Robust and efficient hyperparameter optimization at scale. *arXiv preprint arXiv:1807.01774*.

Gardner, J. R., Kusner, M. J., Xu, Z., Weinberger, K. Q., and Cunningham, J. P. (2014). Bayesian optimization with inequality constraints. In *Proceedings of the 31st International Conference on International Conference on Machine Learning (ICML)*.

Garrido-Merchán, E. C. and Hernández-Lobato, D. (2018). Dealing with categorical and integer-valued variables in bayesian optimization with gaussian processes. *CoRR*.

Gelman, A., Carlin, J. B., Stern, H. S., and Rubin, D. B. (1995). *Bayesian data analysis*. Chapman and Hall/CRC.

González, J. (2016). GPyOpt: A Bayesian optimization framework in Python.

GPy (since 2012). GPy: A gaussian process framework in python. `http://github.com/SheffieldML/GPy`.

Hastie, T. J. (2017). Generalized additive models. In *Statistical models in S*, pages 249–307. Routledge.

Hazan, E., Klivans, A., and Yuan, Y. (2017). Hyperparameter optimization: A spectral approach. In *International Conference on Learning Representations (ICRL)*.

Hennig, P. and Schuler, C. J. (2012). Entropy search for information-efficient global optimization. *JMLR*, 13:1809–1837.

Henrández-Lobato, J. M., Hoffman, M. W., and Ghahramani, Z. (2014). Predictive entropy search for efficient global optimization of black-box functions. In *Proceedings of the 27th International Conference on Neural Information Processing Systems (NIPS)*.

Hernández-Lobato, D., Hernández-Lobato, J. M., and Dupont, P. (2013). Generalized spike-and-slab priors for bayesian group feature selection using expectation propagation. *The Journal of Machine Learning Research*, 14(1):1891–1945.

Hernández-Lobato, J. M., Hernández-Lobato, D., and Suárez, A. (2015). Expectation propagation in linear regression models with spike-and-slab priors. *Machine Learning*, 99(3):437–487.

Hoang, T. N., Hoang, Q. M., Ouyang, R., and Low, K. H. (2018). Decentralized high-dimensional bayesian optimization with factor graphs. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

Hutter, F., Hoos, H. H., and Leyton-Brown, K. (2010). Automated configuration of mixed integer programming solvers. In *International Conference on Integration of Artificial Intelligence (AI) and Operations Research (OR) Techniques in Constraint Programming*.

Hutter, F., Hoos, H. H., and Leyton-Brown, K. (2011). Sequential model-based optimization for general algorithm configuration. In *International Conference on Learning and Intelligent Optimization*, pages 507–523. Springer.

IBM (2009). Users manual for CPLEX. *International Business Machines Corporation*, 46(53):157.

Jenatton, R., Archambeau, C., González, J., and Seeger, M. (2017). Bayesian optimization with tree-structured dependencies. In *International Conference on Machine Learning (ICML)*.

Jones, D. R. (2001). Direct global optimization algorithm. *Encyclopedia of Optimization*.

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Kingma, D. P. and Welling, M. (2013). Auto-encoding variational bayes. In *arXiv preprint arXiv:1312.6114*.

Kirkpatrick, S., Gelatt, C. D., and Vecchi, M. P. (1983). Optimization by simulated annealing. *science*, 220(4598):671–680.

Koller, D., Friedman, N., and Bach, F. (2009). *Probabilistic graphical models: principles and techniques*. MIT press.

Komodakis, N., Paragios, N., and Tziritas, G. (2011). MRF energy minimization and beyond via dual decomposition. *IEEE transactions on pattern analysis and machine intelligence*.

Krause, A., Singh, A., and Guestrin, C. (2008). Near-optimal sensor placements in Gaussian processes: Theory, efficient algorithms and empirical studies. *Journal of Machine Learning Research*.

LeCun, Y., Cortes, C., and Burges, C. (2010). Mnist handwritten digit database. *AT&T Labs [Online]. Available: http://yann. lecun. com/exdb/mnist*, 2:18.

Li, L., Jamieson, K., DeSalvo, G., Rostamizadeh, A., and Talwalkar, A. (2016). Hyperband: A novel bandit-based approach to hyperparameter optimization. *arXiv preprint arXiv:1603.06560*.

Liu, D. C. and Nocedal, J. (1989). On the limited memory BFGS method for large scale optimization. *Mathematical programming*, 45(1-3):503–528.

Minka, T. P. (2001). Expectation propagation for approximate Bayesian inference. In *UAI*, pages 362–369.

Močkus, J. (1975). On Bayesian methods for seeking the extremum. In *Optimization Techniques IFIP Technical Conference*, pages 400–404. Springer.

Mutný, M. and Krause, A. (2018). Efficient high dimensional bayesian optimization with additivity and quadrature fourier features. In *Advances in Neural Information Processing Systems (NIPS)*.

Negoescu, D. M., Frazier, P. I., and Powell, W. B. (2011). The knowledge-gradient algorithm for sequencing experiments in drug discovery. *INFORMS Journal on Computing*.

Nickisch, H. (2012). glm-ie: generalised linear models inference & estimation toolbox. *Journal of Machine Learning Research*, 13(May):1699–1703.

O'Donnell, R. (2014). *Analysis of boolean functions*. Cambridge University Press.

Oh, C., Tomczak, J. M., Gavves, E., and Welling, M. (2019). Combinatorial bayesian optimization using graph representations. *arXiv preprint arXiv:1902.00448*.

Optimization, G. (2014). Gurobi optimizer reference manual. *http://www.gurobi.com*.

Perrone, V., Jenatton, R., Seeger, M., and Archambeau, C. (2017). Multiple adaptive Bayesian linear regression for scalable bayesian optimization with warm start. *arXiv preprint arXiv:1712.02902*.

Rahimi, A. and Recht, B. (2008). Random features for large-scale kernel machines. In *Advances in Neural Information Processing Systems (NIPS)*, pages 1177–1184.

Rahimi, A. and Recht, B. (2009). Weighted sums of random kitchen sinks: Replacing minimization with randomization in learning. In *Advances in Neural Information Processing Systems (NIPS)*.

Rainforth, T., Kosiorek, A. R., Le, T. A., Maddison, C. J., Igl, M., Wood, F., and Teh, Y. W. (2018). Tighter variational bounds are not necessarily better. *arXiv preprint arXiv:1802.04537*.

Rezende, D. J., Mohamed, S., and Wierstra, D. (2014). Stochastic backpropagation and approximate inference in deep generative models. *arXiv preprint arXiv:1401.4082*.

Rolland, P., Scarlett, J., Bogunovic, I., and Cevher, V. (2018). High-dimensional Bayesian optimization via additive models with overlapping groups. *arXiv preprint arXiv:1802.07028*.

Rush, A. M. and Collins, M. (2012). A tutorial on dual decomposition and Lagrangian relaxation for inference in natural language processing. *Journal of Artificial Intelligence Research*, 45:305–362.

Salimans, T., Kingma, D. P., and Welling, M. (2014). Markov chain monte carlo and variational inference: Bridging the gap. *arXiv preprint arXiv:1410.6460*.

Seeger, M. W. (2008). Bayesian inference and optimal design for the sparse linear model. *Journal of Machine Learning Research*, 9(Apr):759–813.

Seeger, M. W. and Nickisch, H. (2008). Compressed sensing and bayesian experimental design. In *International Conference on Machine Learning (ICML)*. ACM.

Seeger, M. W. and Nickisch, H. (2011). Large scale Bayesian inference and experimental design for sparse linear models. *SIAM Journal on Imaging Sciences*, 4(1):166–199.

Shahriari, B., Swersky, K., Wang, Z., Adams, R. P., and de Freitas, N. (2016). Taking the human out of the loop: A review of Bayesian optimization. *Proceedings of the IEEE*.

Snoek, J., Rippel, O., Swersky, K., Kiros, R., Satish, N., Sundaram, N., Patwary, M., Prabhat, M., and Adams, R. (2015). Scalable Bayesian optimization using deep neural networks. In *International Conference on Machine Learning (ICML)*.

Sontag, D., Globerson, A., and Jaakkola, T. (2011). Introduction to dual composition for inference. In *Optimization for Machine Learning*. MIT Press.

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

Sui, Y., Gotovos, A., Burdick, J., and Krause, A. (2015). Safe exploration for optimization with gaussian processes. In *International Conference on Machine Learning*, pages 997–1005.

Thompson, W. R. (1933). On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika*.

Vanschoren, J., Van Rijn, J. N., Bischl, B., and Torgo, L. (2014). Openml: networked science in machine learning. *ACM SIGKDD Explorations Newsletter*, 15(2):49–60.

Wainwright, M. J., Jordan, M. I., et al. (2008). Graphical models, exponential families, and variational inference. *Foundations and Trends® in Machine Learning*, 1(1–2):1–305.

Wang, Z. and Jegelka, S. (2017). Max-value entropy search for efficient bayesian optimization. In *International Conference on Machine Learning (ICML)*.

Williams, C. K. and Rasmussen, C. E. (2006). *Gaussian processes for machine learning*. MIT Press Cambridge, MA.

Xiao, H., Rasul, K., and Vollgraf, R. (2017). Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*.

# A Further Experimental Results

## A.1 Synthetic Benchmark for Unconstrained Optimization

We assess the performance on an unconstrained synthetic linear benchmark function of the form $f(\mathbf{x}) = \mathbf{w}^\top \boldsymbol{\phi}$. We choose a fairly high-dimensional objective with $D_d = 8$ discrete and $D_c = 8$ continuous variables, thus resulting in a total input space dimensionality of $D = D_d + D_c = 16$. For the discrete model part, we choose the $M_d \in \mathcal{O}(D_d^2)$ features $\boldsymbol{\phi}^d$ proposed in Section 3.1. For the continuous features $\boldsymbol{\phi}^c$, we choose $M_c = 16$ dimensional Random Fourier Features to approximate a GP with a squared exponential kernel with bandwidth $\sigma = 1.0$. For the mixed representation, we construct a feature vector $\boldsymbol{\phi}^m$ by stacking all pairs of discrete and continuous features, as proposed in Section 3.1. We consider two settings for the weight vector: Firstly, we sample it from a zero-mean Gaussian, $\mathbf{w} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}) \in \mathbb{R}^M$. Secondly, we sample it from a Laplace distribution, i.e. $\mathbf{w} \sim p(\mathbf{w}|\alpha) \propto \exp(-\alpha^{-1}\|\mathbf{w}\|_1)$, with inverse scale parameter $\alpha = 0.1$, and then prune all weights smaller than 10 to zero to induce sparsity over the weight vector. For the second setting, we also assess MiVaBo using a Laplace prior and the approximate inference technique from (Seeger and Nickisch, 2011).[6] As we do not know the true optimum of the function and thus cannot compute the regret, we normalize all observed function values to the interval $[0, 1]$, resulting in a normalized error as the metric of comparison. We can observe from our results shown in Fig. 4 that MiVaBo outperforms the competing methods in this setting, demonstrating the effectiveness of our approach when its modeling assumptions are fulfilled.

## A.2 Synthetic Benchmark for Constrained Optimization

In another experiment, we demonstrate the capability of our algorithm to incorporate linear constraints on the discrete variables. In particular, we want to enforce a solution that is sparse in the discrete variables via adding a hard cardinality constraint of the type $\sum_{i=1}^{D_d} x_i^d \leq k$, which we can simply specify in the Gurobi optimizer. Cardinality constraints of this type are very relevant in practice, as many real-world problems desire sparse solutions (e.g., sparsification of ising models, contamination control, aero-structural multi-component problems (Baptista and Poloczek, 2018)). We consider the same functional form as before, i.e. again with $D_d = D_c = 8$, and set $k = 2$, meaning that a solution should have at most two of our binary variables set to one, while all others shall be set to zero. To enable comparison with TPE, SMAC and random search, which provide no capability of modeling these kinds of constraints, we assume the objective $f$ to be unconstrained, but instead return a large penalty value if a method acquires an evaluation of $f$ at a point that violates the constraint. Thus, the baseline algorithms are forced to learn the constraint from observations, which is a challenging problem.

One can notice from Fig. 5 that the ability to explicitly encode the cardinality constraint into the discrete optimization oracle significantly increases performance.

[6] We use the MATLAB implementation provided in the `glm-ie` toolbox (Nickisch, 2012) by the same authors.

# B More Details on VAE Hyperparameter Tuning Task

## B.1 Hyperparameters of VAE

We used the `PyTorch` library to implement the VAE used in the experiment. Table 2 describes the names, types and domains of the involved hyperparameters that we tune. Whenever we refer to a "deconvolutional layer" (also called transposed convolution or fractionally-strided convolution), we mean the functional mapping implemented by a `ConvTranspose2d` layer in `PyTorch`[7]. Since our approach operates on a binary encoding of the discrete parameters, we also display the number of bits required to encode each discrete parameter. In total, we consider 25 discrete parameters (resulting in 50 when binarized) as well as three continuous ones.

## B.2 Description of Constraints

We now describe the constraints arising from the mutual dependencies within the hyperparameter space of the deconvolutional VAE (as described in Appendix B.1).

**Encoder constraints.** For the convolutional layers (up to two in our case) of the encoder, we need to ensure that the chosen combination of stride, padding and filter size transforms the input image into an output image whose shape is integral (i.e., not fractional). More precisely, denoting the input image size by $W_{\text{in}}$ (i.e., the input image is quadratic with shape $W_{\text{in}} \times W_{\text{in}}$), the stride by $S$, the filter size by $F$, and the padding by $P$, we need to ensure that the output image size $W_{\text{out}}$ is integral, i.e.

$$W_{\text{out}}^e = (W_{\text{in}}^e - F^e + P^e)/S^e + 1 \in \mathbb{N} \tag{3}$$

where superscripts $e$ are used to make clear that we are considering the encoder. Let us illustrate this with an example[8]: For $W_{\text{in}} = 10$, $P = 0$, $S = 2$ and $F = 3$, we would get an invalid fractional output size of $W_{\text{out}} =$

[7] See `https://pytorch.org/docs/stable/nn.html#convtranspose2d` for details.

[8] This example is taken from `http://cs231n.github.io/convolutional-networks/#conv` (paragraph "Constraints on strides"), which also describes the constraints discussed here. Note that they define the padding $P$ in a slightly different way (i.e., they only consider symmetric padding, while

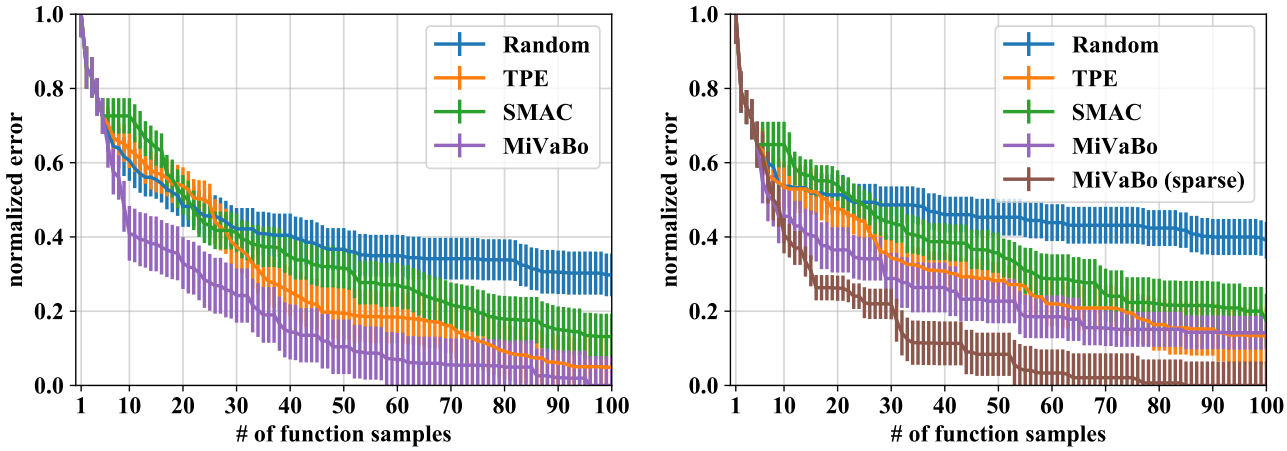


Figure 4: Results on the synthetic benchmark, with the Gaussian (left) and Laplace prior (right). Mean plus/minus one standard deviation of the normalized error over 16 random initializations. (Left) MiVaBo outperforms its competitors. (Right) MiVaBo with a sparse prior outperforms its competitors, including MiVaBo with a Gaussian prior

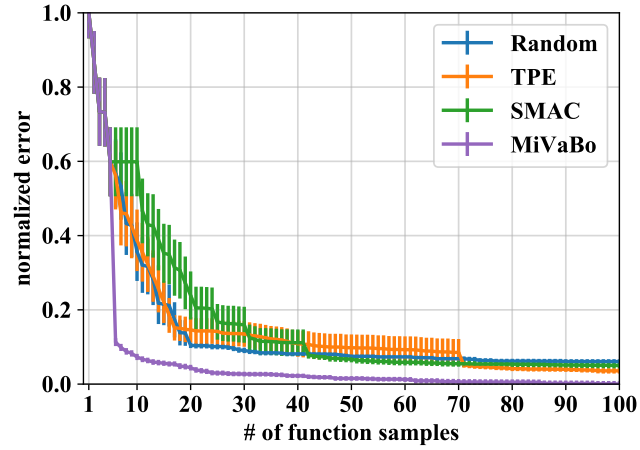


Figure 5: Results on the synthetic benchmark with cardinality constraints. The curves represent the mean plus/minus one standard deviation of the normalized error over 16 random initializations. One can observe that MiVaBo outperforms its competitors.

$(10 - 3 + 0)/2 + 1 = 4.5$. To obtain a valid output size, one could, e.g., instead consider a padding of $P = 1$, yielding $W_{\text{out}} = (10 - 3 + 1)/2 + 1 = 5$. Alternatively, one could also consider a stride of $S = 1$ to obtain $W_{\text{out}} = (10 - 3 + 0)/1 + 1 = 8$, or a filter size of $F = 4$ to obtain $W_{\text{out}} = (10 - 4 + 0)/2 + 1 = 4$ (though the latter is very uncommon and thus not allowed in our setting; we only allow $F \in \{3, 5\}$, as described in Appendix B.1). While this constraint is not trivially fulfilled (which can be verified by manually trying different configurations of $W_{\text{in}}, F, S, P$), it is also not too challenging to find valid configurations.

Note that this constraint is required to be fulfilled for every convolutional layer; we thus obtain the following two constraints in our specific two-layer setting, where $W_{\text{in}} = 28$ (as `MNIST` and `FashionMNIST` images are of shape $28 \times 28$):

$$W^e_{\text{out1}} = (28 - F^e_1 + P^e_1)/S^e_1 + 1 \quad \in \mathbb{N}, \tag{4}$$
$$W^e_{\text{out2}} = (W^e_{\text{out1}} - F^e_2 + P^e_2)/S^e_2 + 1 \quad \in \mathbb{N}. \tag{5}$$

where the subscripts in $\{1, 2\}$ denote the index of the convolutional layer.

Finally, observe that the constraints in Eq. (4) and Eq. (5) are, respectively, linear and quadratic in the discrete variables $F^e_1, F^e_2, P^e_1, P^e_2, S^e_1, S^e_2$, and can thus be readily incorporated into the integer programming solver (e.g. Gurobi (Optimization, 2014) or CPLEX (IBM, 2009)) we employ as a subroutine within our acquisition function optimization strategy.

**Decoder constraints.** While the constraints on the decoder architecture are similar in nature to those for the encoder, they are significantly more difficult to fulfill, which we will now illustrate.

In particular, we need to ensure that the decoder produces images of shape $28 \times 28$. By inverting the formula in Eq. (3), we see that for a deconvolutional layer (which intuitively implements an inversion of the convolution operation), the output image size $W_{\text{out}}$ can be computed as

$$W^d_{\text{out}} = (W^d_{\text{in}} - 1) \times S^d + F^d - 2P^d + O^d \tag{6}$$

where superscripts $d$ are used to make clear that we are considering the decoder, and where $O$ is an additional output padding parameter which can be used to adjust the shape of

we also allow for asymmetric padding) and thus end up with a term of $2P$ instead of $P$ in the formula.

the output image[9]. Note that we now have a factor of $2P$ in Eq. (6) instead of $P$ (as for the encoder, i.e. in Eq. (3)), since we only consider symmetric padding for the decoder, while we allow for asymmetric padding for the encoder (to make it easier to fulfill the integrality constraints for the encoder due to an increased number of valid configurations). The output padding parameter $O$ is required since the mapping from $W^e_{\text{in}}$ to $W^e_{\text{out}}$ in a convolutional layer (i.e. in the encoder) is not bijective: there are different combinations of $W^e_{\text{in}}, F, S, P$ that result in the same $W^e_{\text{out}}$ (which can be easily verified). Thus, given an output size $W^e_{\text{out}}$ (now serving as the input size $W^d_{\text{in}}$ of the deconvolutional layer), there is no unique corresponding input size $W^e_{\text{in}}$ (now serving as the output size $W^d_{\text{out}}$ of the deconvolutional layer). The output padding parameter $O$ can thus be used to disambiguate this relation. Note that $W^d_{\text{out}}$ in Eq. (6) is always integral, so there are no integrality constraints involved here, in constrast to the encoder.

In the context of our decoder model, i.e. with up to two deconvolutional layers, and with a required output image size of 28, we thus obtain the following constraints:

$$W^d_{\text{out}} = (W^d_{\text{in}} - 1) \times S^d_1 + F^d_1 - 2P^d_1 + O^d_1, \quad (7)$$

$$28 = (W^d_{\text{out}} - 1) \times S^d_2 + F^d_2 - 2P^d_2 + O^d_2, \quad (8)$$

i.e. we need to choose the parameters $F^d_1, F^d_2, P^d_1, P^d_2, S^d_1, S^d_2, O^d_1, O^d_2$ such that the output size is 28, which is challenging, as only a small number of parameter configurations fulfill this property. While this problem is already challenging when assuming a given fixed input image shape $W^d_{\text{in}}$, in our setting it is more difficult, as $W^d_{\text{in}}$ has to be of a suitable size as well. Note that $W^d_{\text{in}}$ is determined by the size of the fully-connected layer preceding the first deconvolutional layer, which yields an additional challenge: the size of the last fully-connected layer has to be set such that it can be resized to an image of shape $C^d_1 \times W^d_{\text{in}} \times W^d_{\text{in}}$ (i.e., such that it can be fed into a deconvolutional layer), where $C^d_1$ denotes the number of channels of the first deconvolutional layer of the decoder. As the resulting problem would be too challenging for any algorithm to produce a valid solution in a reasonable amount of time, we simplify it slightly by only treating $C^d_1$ as a design parameter (as described in Appendix B.1), but keeping $W^d_{\text{in}} = 7$ fixed. The value 7 is chosen since $16 \times 7 \times 7 = 784$, i.e., when setting $C^d_1 = 16$, the last fully-connected layer has the correct output shape (since $28 \times 28 = 784$ for an `MNIST` and `FashionMNIST` image). This way, a valid decoder architecture can be achieved by deactivating all convolutional layers and choosing $C^d_1 = 16$, constituting an alternative if fulfilling the decoder constraints in Eq. (7) and Eq. (8) is too challenging for an algorithm.

[9]See e.g. `https://pytorch.org/docs/stable/nn.html#convtranspose2d` for a description of the output padding in the context of the `PyTorch` library we use.

Finally, the constraints in Eq. (7) and Eq. (8) are, respectively, linear and quadratic in the discrete variables $F^d_1, F^d_2, P^d_1, P^d_2, S^d_1, S^d_2, O^d_1, O^d_2$, which again allows us to incorporate them into our optimization routine.

### B.3 Effect of Different Constraint Violation Penalty Values

We now analyze the effect of the constraint violation penalty value on the performance of SMAC, TPE and GPyOpt. Note that random search and MIVABO are not affected by the penalty. We do this analysis to show that the choice of penalty does not qualitatively affect the reported results. In addition to the penalty of 500 nats considered in the experiments in the main paper, we assessed two smaller alternative penalties of 250 nats and 125 nats, respectively. The results in Table 3 show that the performance of the methods improves marginally with decreasing penalty values. This can be intuitively explained by the fact that the smaller the penalty, the smaller the region in hyperparameter space that the penalty discourages from searching. In fact, a large penalty may not only discourage infeasible configurations, but also feasible configurations that lie "close" to the penalized infeasible one (where closeness is defined by the specific surrogate model employed by the method). However, even for the smallest penalty of 125 nats, SMAC, TPE and GPyOpt still perform worse than random search, and thus still significantly worse than MIVABO. Imposing penalties that are significantly smaller than 125 is not sensible, as this will encourage the model-based methods to violate the constraints, and in turn discourage them from ever evaluating a valid configuration (as this would yield a worse score).

Finally, Table 4 shows the number of constraint violations by the different methods, depending on the violation penalty.

### B.4 Visualization of Reconstruction Quality

While log-likelihood scores allow for a principled quantitative comparison between different algorithms, they are typically hard to interpret for humans. We thus in Fig. 6 visualize the reconstruction quality achieved by the best VAE configuration found by the different methods after 32 BO iterations. The VAEs were trained for 32 epochs each (as in the BO experiments). The log-likelihood scores seem to be correlated with quality of visual appearance, and the model found by MIVABO thus may be perceived to produce the visually most appealing reconstructions among all models.

Figure 6: Visualization of the reconstruction quality of a random subset of (non-binarized) images from the MNIST test set, as achieved by the best VAE model (trained for 32 epochs) found by each method. From left to right: ground truth, MIVABO, random search, GPyOpt, TPE and SMAC. The images are thus ordered (from left to right) by increasing negative test log-likelihood achieved by the VAEs used for reconstruction. Interestingly, the log-likelihood seems to capture quality of visual appearance, as the reconstruction quality may be roughly perceived to decrease from left to right.

## C Pseudo-Code for Thompson Sampling

$$p(\mathbf{w}|\mathbf{X}_{1:t}, \mathbf{y}_{1:t}) = \mathcal{N}(\mathbf{m}, \mathbf{S}^{-1}), \quad \text{where}$$
$$\mathbf{m} = \beta \mathbf{S}^{-1} \boldsymbol{\Phi}_{1:t}^\top \mathbf{y}_{1:t} = \beta \mathbf{S}^{-1} (\textstyle\sum_{\tau=1}^t \boldsymbol{\phi}(\mathbf{x}_\tau) y_\tau),$$
$$\mathbf{S} = \alpha \mathbf{I} + \beta \boldsymbol{\Phi}_{1:t}^\top \boldsymbol{\Phi}_{1:t} = \alpha \mathbf{I} + \beta \textstyle\sum_{\tau=1}^t \boldsymbol{\phi}(\mathbf{x}_\tau) \boldsymbol{\phi}(\mathbf{x}_\tau)^\top$$

## D Description of Dual Decomposition

A way to maximize acquisition function in Eq. (2) is via dual decomposition - a powerful approach based on Lagrangian optimization, which has well-studied theoretical properties and has been successfully used for many different problems (Komodakis et al., 2011; Sontag et al., 2011; Rush and Collins, 2012). Despite its versatility, the core idea is simple: decompose the initial problem into smaller solvable subproblems and then extract a solution by cleverly combining the solutions from these subproblems (Komodakis et al., 2011). This requires the following two components: (1) A set of *subproblems* which are defined such that their sum corresponds to the optimization objective, and which can each be optimized globally, and (2) a so-called *master* problem that coordinates the subproblems to find a solution to the original problem. One major advantage of dual decomposition algorithms is that they have well-understood theoretical properties[10], in particular through connections to linear programming (LP) relaxations. In fact, they enjoy the best theoretical guarantees in terms of convergence properties, when compared to other algorithms solving this problem (Komodakis et al., 2011).

**Algorithm 1** THOMPSON SAMPLING

**Require:** model features $\boldsymbol{\phi}(\mathbf{x})$
1: Set $\mathbf{S} = \mathbf{I}$, $\mathbf{m} = \mathbf{0}$
2: **for** $t = 1, 2, \ldots, T$ **do**
3: Sample $\tilde{\mathbf{w}}_t \sim \mathcal{N}(\mathbf{m}, \mathbf{S}^{-1})$
4: Select input $\mathbf{x}_t \in \arg\max_{\mathbf{x} \in \mathcal{X}} \tilde{\mathbf{w}}_t^\top \boldsymbol{\phi}(\mathbf{x})$
5: Query output $y_t = f(\mathbf{x}_t) + \epsilon$
6: Update $\mathbf{S}, \mathbf{m}$ according to Appendix C.
7: **end for**
8: **Output:** $\tilde{\mathbf{x}}_* \in \arg\max_{\mathbf{x} \in \mathcal{X}} \mathbf{m}^\top \boldsymbol{\phi}(\mathbf{x})$

We now describe how to devise a dual decomposition for our problem, by demonstrating how it can be reformulated in terms of master- and sub-problems (see Appendix E for a detailed derivation). For convenience, let us denote the discrete, continuous and mixed parts of Eq. (2) by $f^d(\mathbf{x}^d) = \mathbf{w}^{d\top} \boldsymbol{\phi}^d(\mathbf{x}^d)$, $f^c(\mathbf{x}^c) = \mathbf{w}^{c\top} \boldsymbol{\phi}^c(\mathbf{x}^c)$ and $f^m(\mathbf{x}^d, \mathbf{x}^c) = \mathbf{w}^{m\top} \boldsymbol{\phi}^m(\mathbf{x}^d, \mathbf{x}^c)$, respectively, thus resulting in the representation $f(\mathbf{x}) = f^d(\mathbf{x}^d) + f^c(\mathbf{x}^c) + f^m(\mathbf{x}^d, \mathbf{x}^c)$. First, we note that the discrete $f^d(\mathbf{x}^d)$ and continuous $f^c(\mathbf{x}^c)$ parts of Eq. (2) already represent easy to solve subproblems (as we assume to have access to an optimization oracle). It thus remains to discuss the mixed part $f^m(\mathbf{x}^d, \mathbf{x}^c)$. As $f^m$ is generally difficult to optimize directly, we assume that it decomposes into a sum $f^m(\mathbf{x}) = \sum_{k=1}^{|F|} f_k^m(\mathbf{x}_k^d, \mathbf{x}_k^c)$ of so-called *factors* $f_k^m : \mathcal{X}_k^d \times \mathcal{X}_k^c \to \mathbb{R}$, where $\mathbf{x}_k^d \in \mathcal{X}_k^d$ and $\mathbf{x}_k^c \in \mathcal{X}_k^c$ respectively denote subvectors of $\mathbf{x}^d$ and $\mathbf{x}^c$ from the (typically low-dimensional) subspaces $\mathcal{X}_k^d \subseteq \mathcal{X}^d$ and $\mathcal{X}_k^c \subseteq \mathcal{X}^c$. Here, $F$ denotes a set of subsets $k \in F$ of the variables. Given this formulation, the initial problem then reduces[11] to the minimization of the dual function $L(\boldsymbol{\lambda})$ w.r.t. Lagrange multipliers $\boldsymbol{\lambda}$, i.e., the master problem $\min_{\boldsymbol{\lambda}} L(\boldsymbol{\lambda})$, with dual function $L(\boldsymbol{\lambda}) = \max_{\mathbf{x}^d} \left\{ f^d(\mathbf{x}^d) + \sum_{k \in F} \boldsymbol{\lambda}_k^d \mathbf{x}_{|k}^d \right\} + \max_{\mathbf{x}^c} \left\{ f^c(\mathbf{x}^c) + \right.$

[10] For details, we refer the interested reader to (Komodakis et al., 2011; Sontag et al., 2011; Rush and Collins, 2012)

[11] Refer to Appendix E for a detailed derivation.

$\sum_{k\in F} \boldsymbol{\lambda}_k^c \mathbf{x}_{|k}^c\} + \sum_{k\in F} \max_{\mathbf{x}_k^d, \mathbf{x}_k^c} \{f_k^m(\mathbf{x}_k^d, \mathbf{x}_k^c) - \boldsymbol{\lambda}_k^d \mathbf{x}_k^d - \boldsymbol{\lambda}_k^c \mathbf{x}_k^c\}$. Here, the master problem coordinates the $2 + |F|$ maximization subproblems, where $\mathbf{x}_{|k}^d$and $\mathbf{x}_{|k}^c$ respectively denote the subvectors of $\mathbf{x}^d$ and $\mathbf{x}^c$ containing only the variables of factor $k \in F$, $\boldsymbol{\lambda}_k^d$ and $\boldsymbol{\lambda}_k^c$ are their corresponding Lagrange multipliers. Intuitively, by updating the dual variables $\boldsymbol{\lambda}$, the master problem ensures agreement on the involved variables between the discrete and continuous subproblems and the mixed factors. Importantly, the dual function $L(\boldsymbol{\lambda})$ only involves independent maximization over local assignments of $\mathbf{x}^d, \mathbf{x}^c$ and $\mathbf{x}_k^d, \mathbf{x}_k^c$, which are assumed to be tractable. There are two main classes of algorithms used for the maximization, namely subgradient methods and block coordinate descent (Sontag et al., 2011).

# E Derivation of Dual Decomposition

One interesting interpretation of our acquisition function optimization problem as defined in Section 3.3 is as *maximum a posteriori* (MAP) inference in the undirected graphical model, or Markov random field (MRF) (Koller et al., 2009), induced by the dependency graph of the involved variables (i.e. the graph in which vertices correspond to variables, and edges appear between variables that interact in some way). We take this perspective and devise a dual decomposition to tackle the MAP estimation problem induced by our particular setting (i.e., interpreting our acquisition function as the energy function of the graphical model), following the formulation of (Sontag et al., 2011).[12]

Consider a graphical model on the vertex set $\mathcal{V} = V_d \cup V_c$, where the vertices $V_d = \{1, \dots, D_d\}$ and $V_c = \{D_d + 1, \dots, D_d + D_c\}$ correspond to the discrete and continuous variables $\mathbf{x}^d \in \mathcal{X}^d$ and $\mathbf{x}^c \in \mathcal{X}^c$, respectively. Furthermore, consider a set $F$ of subsets of both discrete and continuous variables/vertices, i.e., $\forall f \in F : f = (f^d \cup f^c) \subseteq V, \emptyset \neq f^d \subseteq V_d, \emptyset \neq f^c \subseteq V_c$, where each subset corresponds to the domain of one of the factors.

Now assume that we are given the following functions on these factors as well as on all discrete/continuous variables:

- A factor $\theta^d(\mathbf{x}^d), \theta^d : \mathcal{X}^d \to \mathbb{R}$ on all discrete variables
- A factor $\theta^c(\mathbf{x}^c), \theta^c : \mathcal{X}^c \to \mathbb{R}$ on all continuous variables
- $|F|$ mixed factors $\theta_f^m(\mathbf{x}_f^d, \mathbf{x}_f^c), \theta_f^m : \mathcal{X}_f^d \times \mathcal{X}_f^c \to \mathbb{R}$ on subsets $f \in F$ of both discrete and continuous variables, where $\mathbf{x}_f^d \in \mathcal{X}_f^d$ and $\mathbf{x}_f^c \in \mathcal{X}_f^c$ respectively denote subvectors of $\mathbf{x}^d$ and $\mathbf{x}^c$ from the (typically low-dimensional) subspaces $\mathcal{X}_f^d \subseteq \mathcal{X}^d$ and $\mathcal{X}_f^c \subseteq \mathcal{X}^c$, indexed by the vertices contained in $f$

The goal of our MAP problem is to find and assignment to all variables $\mathbf{x}^d$ and $\mathbf{x}^c$ which maximizes the sum of the factors:

$$\text{MAP}(\boldsymbol{\theta}) = \max_{\mathbf{x}} \left\{ \theta^d(\mathbf{x}^d) + \theta^c(\mathbf{x}^c) + \sum_{f\in F} \theta_f^m(\mathbf{x}_f^d, \mathbf{x}_f^c) \right\} \tag{9}$$

We now slightly reformulate this problem by duplicating the variables $x_i^d$ and $x_j^c$, once for each mixed factor $\theta_f^m(\mathbf{x}_f^d, \mathbf{x}_f^c)$, and then enforce that these variables are equal to the ones appearing in the factors $\theta^d(\mathbf{x}^d)$ and $\theta^c(\mathbf{x}^c)$, respectively. Let $x_i^{df}$ and $x_j^{cf}$ respectively denote the copy of $x_i^d$ and $x_j^c$ used by factor $f$. Moreover, denote by $\mathbf{x}_f^{df} = \{x_i^{df}\}_{i\in f^d}$ and $\mathbf{x}_f^{cf} = \{x_j^{cf}\}_{j\in f^c}$ the set of variables used by factor $f$, and by $\mathbf{x}^F = \{\mathbf{x}_f^{df}, \mathbf{x}_f^{cf}\}_{f\in F}$ the set of all variable copies. We then get the equivalent (but now constrained) optimization problem

$$\max_{\mathbf{x}, \mathbf{x}^F} \left\{ \theta^d(\mathbf{x}^d) + \theta^c(\mathbf{x}^c) + \sum_{f\in F} \theta_f^m(\mathbf{x}_f^{df}, \mathbf{x}_f^{cf}) \right\} \tag{10}$$

$$\text{s.t.} \quad x_i^{df} = x_i^d, \quad \forall f \in F, i \in f^d$$
$$x_j^{cf} = x_j^c, \quad \forall f \in F, j \in f^c$$

To remove the coupling constraints, (Sontag et al., 2011) now propose to use the technique of *Lagrangian relaxation* and introduce a Lagrange multiplier / dual variable $\lambda_{fi}(x_i)$ for every choice of $f \in F$, $i \in f$ and $x_i$ (i.e. for every factor, for every variable in that factor, and for every value of that variable). These multipliers may then be interpreted as the *message* that factor $f$ sends to variable $i$ about its state $x_i$.

While this works well if all variables are discrete, in our model we also have continuous variables $x_j^c$, and it is clearly not possible to have a Lagrange multiplier for every possible value of $x_j^c$. To mitigate this issue, we follow (Komodakis et al., 2011) and instead only introduce a multiplier $\lambda_{fi}$ for every choice of $f \in F$ and $i \in f$, and model the interaction with the variables as $\lambda_{fi}(x_i) = \lambda_{fi} x_i$ (i.e., the product of a multiplier $\lambda_{fi}$ and variable $x_i$). Observe that since our goal is to relax the coupling constraints, it is sufficient to introduce one multiplier per constraint. Since we have a constraint for every factor $f \in F$ and every discrete variable $i \in f^d$ and continuous variable $j \in f^c$ in that factor, our approach is clearly viable.

Note that in contrast to (Komodakis et al., 2011), that introduces a set of multipliers for *every* factor / subgraph, we only introduce multipliers for the *mixed* factors $f \in F$. This is because in contrast to (Komodakis et al., 2011), we do not introduce a full set of variable copies for *every* factor and then couple them to another global set of "original"

[12] In accordance with the notation in (Sontag et al., 2011), we will here denote the factors by $\theta$ instead of $f$ (i.e., in contrast to the main text).

variables, but we instead only introduce variable copies for the *mixed* factors and couple them to the variables appearing in the *discrete and continuous* factors, which we assume to be the "original" variables instead. This essentially is the same approach used in (Sontag et al., 2011), with the difference that (Sontag et al., 2011) introduce a singleton factor for each variable (i.e., a factor which depends only on a single variable), which they consider to be the "original" variable. They then simply couple the variable copies appearing in the *higher-order* factors to the "original" variables appearing in the *singleton* factors. In contrast, in our formulation we don't introduce singleton factors to model the "original" variables, but instead use the *fully discrete and continuous* factors for this purpose, which clearly works equally well. Note that as a result of this modeling choice, our optimization problem will be unconstrained, regardless of the number of factors, similar as in (Sontag et al., 2011). In contrast, (Komodakis et al., 2011) end up with constraints enforcing that some of the dual variables sum to zero, since they are optimizing out the global set of "original" variables from their objective, while we keep the set of "original" variables within our discrete and continuous factors. For this reason, we will in contrast to (Komodakis et al., 2011) later not require a projection step within the subgradient method used to optimize the dual; this is to be detailed further below.

For clarity, we treat discrete and continuous variables distinctly and for factor $f \in F$ denote $\lambda^d_{fi}$ and $\lambda^c_{fj}$ respectively for the Lagrange multipliers corresponding to its discrete variables $i \in f^d$ (or rather, the constraints $x_i^{df} = x_i^d$) and its continuous variables $j \in f^c$ (or rather, the constraints $x_j^{cf} = x_j^c$). For every factor $f \in F$, we furthermore aggregate its multipliers into the vectors $\boldsymbol{\lambda}^d_f = \{\lambda^d_{fi}\}_{i \in f^d} \in \mathbb{R}^{|\mathcal{X}^d_f|}$ and $\boldsymbol{\lambda}^c_f = \{\lambda^c_{fj}\}_{j \in f^c} \in \mathbb{R}^{|\mathcal{X}^c_f|}$. The set of all Lagrange multipliers is thus $\boldsymbol{\lambda} = \{\lambda^d_{fi} : f \in F, i \in f^d\} \cup \{\lambda^c_{fj} : f \in F, j \in f^c\} = \{\boldsymbol{\lambda}^d_f, \boldsymbol{\lambda}^c_f\}_{f \in F}$. We then define the Lagrangian

$$
\begin{aligned}
L(\boldsymbol{\lambda}, \mathbf{x}, \mathbf{x}^F) &= \theta^d(\mathbf{x}^d) + \theta^c(\mathbf{x}^c) + \sum_{f \in F} \theta^m_f(\mathbf{x}^{df}_f, \mathbf{x}^{cf}_f) \\
&+ \sum_{f \in F} \sum_{i \in f^d} \lambda^d_{fi} \left( x^d_i - x^{df}_i \right) + \sum_{f \in F} \sum_{j \in f^c} \lambda^c_{fj} \left( x^c_j - x^{cf}_j \right) \\
&= \left( \theta^d(\mathbf{x}^d) + \sum_{f \in F} \sum_{i \in f^d} \lambda^d_{fi} x^d_i \right) \\
&+ \left( \theta^c(\mathbf{x}^c) + \sum_{f \in F} \sum_{j \in f^c} \lambda^c_{fj} x^c_j \right) \\
&+ \sum_{f \in F} \left( \theta^m_f(\mathbf{x}^{df}_f, \mathbf{x}^{cf}_f) - \sum_{i \in f^d} \lambda^d_{fi} x^{df}_i - \sum_{j \in f^c} \lambda^c_{fj} x^{cf}_j \right) .
\end{aligned}
$$

This results in the following optimization problem:

$$
\begin{aligned}
&\max_{\mathbf{x}, \mathbf{x}^F} L(\boldsymbol{\lambda}, \mathbf{x}, \mathbf{x}^F) \\
\text{s.t.} \quad & x^{df}_i = x^d_i, \quad \forall f \in F, i \in f^d \\
& x^{cf}_j = x^c_j, \quad \forall f \in F, j \in f^c
\end{aligned} \tag{11}
$$

Note that the problem in Eq. (11) is still equivalent to our (hard) original problem in Eq. (9) for any assignment of $\boldsymbol{\lambda}$, since the Lagrange multipliers cancel out if all coupling constraints are fulfilled.

To obtain a tractable problem, we thus simply omit the coupling constraints in Eq. (11) and define the dual function $L(\boldsymbol{\lambda})$ as

$$
\begin{aligned}
L(\boldsymbol{\lambda}) &= \max_{\mathbf{x}, \mathbf{x}^F} L(\boldsymbol{\lambda}, \mathbf{x}, \mathbf{x}^F) \\
&= \max_{\mathbf{x}^d} \left( \theta^d(\mathbf{x}^d) + \sum_{f \in F} \sum_{i \in f^d} \lambda^d_{fi} x^d_i \right) \\
&+ \max_{\mathbf{x}^c} \left( \theta^c(\mathbf{x}^c) + \sum_{f \in F} \sum_{j \in f^c} \lambda^c_{fj} x^c_j \right) \\
&+ \sum_{f \in F} \max_{\mathbf{x}^{df}_f, \mathbf{x}^{cf}_f} \left( \theta^m_f(\mathbf{x}^{df}_f, \mathbf{x}^{cf}_f) - \sum_{i \in f^d} \lambda^d_{fi} x^{df}_i - \sum_{j \in f^c} \lambda^c_{fj} x^{cf}_j \right)
\end{aligned}
$$

Note that the maximizations are now fully independent, such that we can (without introducing any ambiguity) simplify the notation for the variables involved in the mixed terms to denote $\mathbf{x}^d_f$ and $\mathbf{x}^c_f$ instead of $\mathbf{x}^{df}_f$ and $\mathbf{x}^{cf}_f$, respectively[13], resulting in the slightly simpler dual formulation

$$
\begin{aligned}
L(\boldsymbol{\lambda}) &= \max_{\mathbf{x}^d} \left( \theta^d(\mathbf{x}^d) + \sum_{f \in F} \sum_{i \in f^d} \lambda^d_{fi} x^d_i \right) \\
&+ \max_{\mathbf{x}^c} \left( \theta^c(\mathbf{x}^c) + \sum_{f \in F} \sum_{j \in f^c} \lambda^c_{fj} x^c_j \right) \\
&+ \sum_{f \in F} \max_{\mathbf{x}^d_f, \mathbf{x}^c_f} \left( \theta^m_f(\mathbf{x}^d_f, \mathbf{x}^c_f) - \sum_{i \in f^d} \lambda^d_{fi} x^d_i - \sum_{j \in f^c} \lambda^c_{fj} x^c_j \right)
\end{aligned}
$$

Let $\mathbf{x}^d_{|f} \in \mathcal{X}^d_f$ and $\mathbf{x}^c_{|f} \in \mathcal{X}^c_f$ respectively denote the subvectors of $\mathbf{x}^d$ and $\mathbf{x}^c$ containing only the variables of factor $f$. The shorthands (or *reparameterizations* (Sontag et al.,

[13] I.e., we replace all variable copies $\mathbf{x}^{df}_f, \mathbf{x}^{cf}_f$ in the mixed terms by the "original" variables $\mathbf{x}^d_f, \mathbf{x}^c_f$.

2011))

$$\bar{\theta}_d^{\boldsymbol{\lambda}}(\mathbf{x}^d) = \theta^d(\mathbf{x}^d) + \sum_{f\in F}\sum_{i\in f^d} \lambda_{fi}^d x_i^d$$
$$= \theta^d(\mathbf{x}^d) + \sum_{f\in F} \boldsymbol{\lambda}_f^d \mathbf{x}_{|f}^d \tag{12}$$
$$\bar{\theta}_c^{\boldsymbol{\lambda}}(\mathbf{x}^c) = \theta^c(\mathbf{x}^c) + \sum_{f\in F}\sum_{j\in f^c} \lambda_{fj}^c x_j^c$$
$$= \theta^c(\mathbf{x}^c) + \sum_{f\in F} \boldsymbol{\lambda}_f^c \mathbf{x}_{|f}^c \tag{13}$$
$$\bar{\theta}_f^{\boldsymbol{\lambda}}(\mathbf{x}_f^d, \mathbf{x}_f^c) = \theta_f^m(\mathbf{x}_f^d, \mathbf{x}_f^c) - \sum_{i\in f^d} \lambda_{fi}^d x_i^d - \sum_{j\in f^c} \lambda_{fj}^c x_j^c$$
$$= \theta_f^m(\mathbf{x}_f^d, \mathbf{x}_f^c) - \boldsymbol{\lambda}_f^d \mathbf{x}_f^d - \boldsymbol{\lambda}_f^c \mathbf{x}_f^c \tag{14}$$

further simplify the dual function $L(\boldsymbol{\lambda})$ to

$$L(\boldsymbol{\lambda}) = \max_{\mathbf{x}^d} \bar{\theta}_d^{\boldsymbol{\lambda}}(\mathbf{x}^d) + \max_{\mathbf{x}^c} \bar{\theta}_c^{\boldsymbol{\lambda}}(\mathbf{x}^c) + \sum_{f\in F} \max_{\mathbf{x}_f^d, \mathbf{x}_f^c} \bar{\theta}_f^{\boldsymbol{\lambda}}(\mathbf{x}_f^d, \mathbf{x}_f^c)\,. \tag{15}$$

First, observe that since we maximize over $\mathbf{x}$ and $\mathbf{x}^F$, the dual function $L(\boldsymbol{\lambda})$ is a function of just the Lagrange multipliers $\boldsymbol{\lambda}$. Note that since $L(\boldsymbol{\lambda})$ maximizes over a larger space (since instead of forcing that there must be one global assignment maximizing the objective, we allow the discrete/continuous potentials to be maximized independently of the mixed potentials, meaning that $\mathbf{x}$ may not coincide with $\mathbf{x}^F$), we have for all $\boldsymbol{\lambda}$ that

$$\text{MAP}(\boldsymbol{\theta}) \leq L(\boldsymbol{\lambda})\,. \tag{16}$$

The *dual problem* now is to find the tightest upper bound by optimizing the Lagrange multipliers, i.e.

$$\min_{\boldsymbol{\lambda}} L(\boldsymbol{\lambda}) \tag{17}$$

We also call the dual problem in Eq. (17) the *master problem*, which coordinates the $2+|F|$ *slave problems* (i.e., one for each factor)

$$s^d(\boldsymbol{\lambda}) = \max_{\mathbf{x}^d} \bar{\theta}_d^{\boldsymbol{\lambda}}(\mathbf{x}^d) \tag{18a}$$
$$s^c(\boldsymbol{\lambda}) = \max_{\mathbf{x}^c} \bar{\theta}_c^{\boldsymbol{\lambda}}(\mathbf{x}^c) \tag{18b}$$
$$s^f(\boldsymbol{\lambda}) = \max_{\mathbf{x}_f^d, \mathbf{x}_f^c} \bar{\theta}_f^{\boldsymbol{\lambda}}(\mathbf{x}_f^d, \mathbf{x}_f^c), \quad \forall f \in F\,. \tag{18c}$$

where we refer to $s^d$, $s^c$ and $s^f$ as the discrete slave, the continuous slave, and the mixed slaves, respectively. Using the notation in Eqs. (18a)-(18c), the dual function further simplifies to

$$L(\boldsymbol{\lambda}) = s^d(\boldsymbol{\lambda}) + s^c(\boldsymbol{\lambda}) + \sum_{f\in F} s^f(\boldsymbol{\lambda})\,. \tag{19}$$

Intuitively, the goal of Eq. (17) is as follows: The master problem wants the discrete/continuous slaves to agree with the mixed slaves/factors in which the corresponding discrete/continuous variables appear, and conversely, it wants the mixed slaves to agree with the slaves/factors of the discrete/continuous variables in its scope. The master problem will thus incentivize the discrete/continuous slaves and the mixed slaves to agree with each other, which is done by updating the dual variables $\boldsymbol{\lambda}$ accordingly.

The key property of the function $L(\boldsymbol{\lambda})$ is that it only involves maximization over local assignments of $\mathbf{x}^d, \mathbf{x}^c$ and $\mathbf{x}_f^d, \mathbf{x}_f^c$, which are tasks we assume to be tractable. The dual thus decouples the original problem, resulting in a problem that can be optimized using local operations. Algorithms that minimize the approximate objective $L(\boldsymbol{\lambda})$ use local updates where each iteration of the algorithms repeatedly finds a maximizing assignment for the subproblems individually, using these to update the dual variables $\boldsymbol{\lambda}$ that glue the subproblems together. There are two main classes of algorithms of this kind, one based on a subgradient method and another based on block coordinate descent (Sontag et al., 2011).

# F Details on Baseline Implementations and XGBoost Benchmark

We use the publicly available `Python` implementations of SMAC (Hutter et al., 2011) (`https://github.com/automl/SMAC3`), TPE (Bergstra et al., 2011b) (`https://github.com/hyperopt/hyperopt`), and GPyOpt (González, 2016) (`https://github.com/SheffieldML/GPyOpt`). For the GP with simulated annealing baseline, we use GPy (GPy, 2012) and the simulated annealing implementation at `https://github.com/perrygeo/simanneal`. We use the default values provided by these packages for any hyperparameters, unless stated otherwise.

Furthermore, please refer to the corresponding websites for details on the OpenML XGBoost benchmark (`https://www.openml.org/f/6767`), on the underlying implementation (`https://www.rdocumentation.org/packages/xgboost/versions/0.6-4`), and on the `steel-plates-fault` (`https://www.openml.org/t/9967`) and `monks-problem-1` (`https://www.openml.org/t/146064`) datasets.

Finally, see Table 5 for a description of the hyperparameters involved in XGBoost.

Table 2: **Hyperparameters of the VAE.** The architecture of the VAE (if all layers are enabled) is `C1-C2-F1-F2-z-F3-F4-D1-D2`, with `C` denoting a convolutional (conv.) layer, `F` a fully-connected (fc.) layer, `D` a deconvolutional (deconv.) layer and `z` the latent space. Layers `F2` and `F3` have fixed sizes of $2d_z$ and $d_z$ units respectively, where $d_z$ denotes the dimensionality of the latent space `z`. The domain of the number of units of the fc. layers `F1` and `F4` is discretized with a step size of 64, i.e. $[0, 64, 128, \ldots, 832, 896, 960]$, denoted by $[0 \ldots 960]$ in the table for brevity. For $d_z$, the domain $[16 \ldots 64]$ refers to all integers within that interval.

| # | Name | Type | Domain | Bits |
|---|---|---|---|---|
| 1 | Number of conv. layers in encoder | discrete | [0,1,2] | 2 |
| | Parameters of `C1` | | | |
| 2 | Number of channels of `C1` | discrete | [4,8,16,24] | 2 |
| 3 | Stride of `C1` | discrete | [1,2] | 1 |
| 4 | Filter size of `C1` | discrete | [3,5] | 1 |
| 5 | Padding of `C1` | discrete | [0,1,2,3] | 2 |
| | Parameters of `C2` | | | |
| 6 | Number of channels of `C2` | discrete | [8,16,32,48] | 2 |
| 7 | Stride of `C2` | discrete | [1,2] | 1 |
| 8 | Filter size of `C2` | discrete | [3,5] | 1 |
| 9 | Padding of `C2` | discrete | [0,1,2,3] | 2 |
| 10 | Number of fc. layers in encoder | discrete | [0,1,2] | 2 |
| 11 | Number of units of `F1` | discrete | [0...960] | 4 |
| 12 | Dimensionality $d_z$ of `z` | discrete | [16...64] | 6 |
| 13 | Number of fc. layers in decoder | discrete | [0,1,2] | 2 |
| 14 | Number of units of `F4` | discrete | [0...960] | 4 |
| 15 | Number of deconv. layers in decoder | discrete | [0,1,2] | 2 |
| | Parameters of `D1` | | | |
| 16 | Number of channels of `D1` | discrete | [8,16,32,48] | 2 |
| 17 | Stride of `D1` | discrete | [1,2] | 1 |
| 18 | Filter size of `D1` | discrete | [3,5] | 1 |
| 19 | Padding of `D1` | discrete | [0,1,2,3] | 2 |
| 20 | Output padding of `D1` | discrete | [0,1,2,3] | 2 |
| | Parameters of `D2` | | | |
| 21 | Number of channels of `D2` | discrete | [4,8,16,24] | 2 |
| 22 | Stride of `D2` | discrete | [1,2] | 1 |
| 23 | Filter size of `D2` | discrete | [3,5] | 1 |
| 24 | Padding of `D2` | discrete | [0,1,2,3] | 2 |
| 25 | Output padding of `D2` | discrete | [0,1,2,3] | 2 |
| 26 | Learning rate | continuous | $[10^{-4}, 10^{-2}]$ | - |
| 27 | Learning rate decay factor | continuous | $[0.5, 1.0]$ | - |
| 28 | Weight decay regularization | continuous | $[10^{-6}, 10^{-2}]$ | - |
| | **Total** | | | **50** |

Table 3: Mean plus/minus one standard deviation of the negative test log-likelihood over 8 random initializations, achieved by the best VAE configuration found by SMAC, TPE and GPyOpt after 16 BO iterations, for constraint violation penalties of 500, 250 and 125 nats. Performance values of MIVABO and random search (which are not affected by the penalty) are included for reference.

| **Algorithm** | **Penalty (nats)** | | |
|---|---|---|---|
| | **500** | **250** | **125** |
| **SMAC** | $113.0 \pm 1.8$ | $112.1 \pm 1.8$ | $111.1 \pm 1.6$ |
| **TPE** | $108.8 \pm 1.2$ | $108.1 \pm 1.3$ | $108.1 \pm 1.3$ |
| **GPyOpt** | $108.5 \pm 1.1$ | $108.5 \pm 0.6$ | $106.5 \pm 1.4$ |
| **RS** | | | $106.3 \pm 0.9$ |
| **MIVABO** | | | $\mathbf{94.4 \pm 0.8}$ |

Table 4: Mean plus/minus one standard deviation of the number of constraint violations by SMAC, TPE, GPyOpt and random search within 16 BO iterations over 8 random initializations, for constraint violation penalties of 500, 250 and 125 nats.

| **Algorithm** | **Penalty (nats)** | | |
|---|---|---|---|
| | **500** | **250** | **125** |
| **SMAC** | $37 \pm 21.7$ | $36 \pm 21.9$ | $28 \pm 11.6$ |
| **TPE** | $67 \pm 21.3$ | $68 \pm 22.2$ | $68 \pm 22.2$ |
| **GPyOpt** | $36 \pm 19.3$ | $32 \pm 18.0$ | $27 \pm 10.4$ |
| **Random search** | $71 \pm 25.5$ | $71 \pm 25.5$ | $71 \pm 25.5$ |

Table 5: Hyperparameters of the XGBoost algorithm. 10 parameters, 3 of which are discrete.

| **Name** | **Type** | **Domain** |
|---|---|---|
| booster | discrete | ['gbtree', 'gblinear'] |
| nrounds | discrete | $[3, 5000]$ |
| alpha | continuous | $[0.000985, 1009.209690]$ |
| lambda | continuous | $[0.000978, 999.020893]$ |
| colsample_bylevel | continuous | $[0.046776, 0.998424]$ |
| colsample_bytree | continuous | $[0.062528, 0.999640]$ |
| eta | continuous | $[0.000979, 0.995686]$ |
| max_depth | discrete | $[1, 15]$ |
| min_child_weight | continuous | $[1.012169, 127.041806]$ |
| subsample | continuous | $[0.100215, 0.999830]$ |