d-MALIBOO: a Bayesian Optimization framework for dealing with Discrete Variables

Roberto Sala*, Bruno Guindani*, Danilo Ardagna* and Alessandra Guglielmi†
* *Department of Electronics, Information and Bioengineering, Politecnico di Milano, Milan, Italy*
† *Department of Mathematics, Politecnico di Milano, Milan, Italy*
Email: name.surname@polimi.it

***Abstract*—Cloud computing has become essential for delivering flexible and scalable resources. In this environment, finding the optimal hardware-software configuration is crucial and often involves solving constrained black-box problems on discrete multidimensional domains. These optimizations must be performed within a limited number of evaluations to contain costs. Bayesian Optimization (BO) addresses this problem by providing near-optimal solutions with few iterations. However, the original BO algorithm was designed for continuous and unbounded domains. On the other hand, Machine Learning (ML) models are powerful tools for predicting the values of a black-box function. In this work, we present *d-MALIBOO*, a framework that integrates BO and ML techniques to improve the efficiency of finding optimal solutions in discrete and bounded domains. While BO suggests the next point to be evaluated by effectively balancing exploration and exploitation, ML models are valuable for determining feasibility regions and focusing the search for the optimum. Experimental results show that our algorithm outperforms alternative methods from the literature in all tested environments, especially in complex optimization scenarios, resulting in an improvement in regret by approximately 2–8 times.**

***Index Terms*—Discrete Variables, Bayesian Optimization, Machine Learning, Black-box Optimization, Cloud Computing**

I. Introduction

In the era of cloud computing, finding the optimal configuration for both infrastructure and application parameters has become increasingly crucial. Cloud computing provides flexible and scalable resources, but optimizing these resources presents significant challenges. In cloud environments, identifying the best hardware-software configuration often involves constrained problems on discrete and multidimensional domains. These optimization processes must be performed within a limited number of iterations. For example, determining the optimal hardware settings for computational resources is essential in Artificial Intelligence (AI) applications to minimize costs while maintaining Quality of Service (QoS) constraints. Additionally, selecting the optimal software parameter values for these models is critical to achieving high prediction accuracy within time constraints [1]. Bayesian Optimization (BO) has gained attention as an effective method for addressing global optimization problems involving expensive-to-evaluate black-box functions [2]. BO operates as a sequential approach, requiring minimal iterations to converge to near-optimal solutions. It selects new evaluation points that strike a balance between exploration (high uncertainty) and exploitation (high expected value). BO is suitable for continuous variables, unbounded domains, and continuous objective functions (OF). However, in the scenarios described earlier, the domains of the decision variables that determine the execution configuration are often discrete and bounded. This setting can lead to less efficient convergence toward the optimal solution.

In [3], the authors introduce MALIBOO (MAchine Learning In Bayesian OptimizatiOn), a framework that combines BO with ML techniques to lower the execution costs of recurring resource-constrained computing tasks. It presents a comprehensive theoretical framework suitable for diverse optimization problems, including those with black-box constraints and OFs. The framework is especially beneficial for cloud optimization scenarios, where evaluating configurations can be costly. Although this framework is not specifically designed to handle discrete variables, it implements the possibility of applying BO over discrete and bounded domains. Special attention is given to preventing the repetition of evaluating points already assessed. However, MALIBOO has been developed to primarily work with continuous variables.

In this work, we present *d-MALIBOO*, a BO-based algorithm for solving constrained optimization problems on discrete domains. Building on the original MALIBOO: (1) We implement several novel ML-based approaches for estimating black-box constraints and for improving the search for the optimal value; we also propose an ε-greedy approach to navigate the exploration-exploitation trade-off. (2) We test the proposed approaches on different applications ranging from AI to edge and high-performance computing (HPC) domains. Finally, (3) we compare our techniques with two competitors from the literature. Experimental results show that our new algorithms are more efficient than the original MALIBOO, especially in more complex scenarios, and outperform competitor methods in all tested settings. Our approach often leads to a 2- to 8-fold reduction in regret compared to state-of-the-art methods.

This work is structured as follows. Section II shows the mathematical formulation of the problem of interest, while Section III provides an overview of BO-based techniques. Section IV describes our contributions to dealing with discrete optimization problems. Experimental results are presented in Section V. Section VI discusses related works. Finally, Section VII provides conclusions and proposes future developments.

II. Problem Overview

In this work, we address a constrained optimization problem with the following mathematical formulation:

$$\begin{aligned} &\min_{x \in D} f(x) + \eta \\ &\text{s.t. } g_j(x) \in [G^j_{min}, G^j_{max}],\ j = 1, \ldots, C \end{aligned} \tag{1}$$

where $f(x)$ is the OF to minimize, η is a noise measurement error on the OF, $D \subset \mathbb{R}^d$ is a d-dimensional discrete domain, and C is the total number of constraints. The constrained functions $g_j(x)$ are generally black-box and can be either dependent or independent of $f(x)$, while x is the input vector of configurations, also referred to as parameters or features. Typically, $d \leq 20$ and the OF $f(x)$ is continuous, black-box, expensive to evaluate, and its derivative(s) are not known [2]. For instance, $f(x)$ is an accuracy metric or cost for running a Deep Learning Model on a cloud cluster and $g(x)$ is the execution time on which we place an upper bound.

The original BO algorithm usually deals with optimization problems in unconstrained domains. However, in this study, we aim to challenge the assumption of continuity of the optimization domain and instead explore discrete, bounded domains. These characteristics significantly increase the complexity of the optimization problem, though they model many realistic scenarios. Consider, for example, the hyperparameter tuning problem for an HPC system, for which the features are the number of cores $(1, 2, 4, 8 \ldots)$, the memory size $(256, 512, 1024 \ldots$ MB), the number of nodes used, etc.; we address such scenarios in our experimental analysis. In these cases, we are not considering all possible integer values within an interval for each feature, nor are these values equispaced. This makes approaches such as [4] not applicable, as it assumes that discrete values are equispaced.

III. Bayesian Optimization Background

This section presents the classical BO (Section III-A) and the MALIBOO algorithm (Section III-B) proposed in [3].

A. The Bayesian Optimization algorithm

BO is a powerful method for optimizing black-box OFs, since it can yield near-optimal solutions in a relatively small number of iterations. This is particularly relevant in scenarios where each evaluation of the OF incurs significant costs or requires substantial execution time, as mentioned in Section II. In such cases, conducting an extensive search over the domain is not economically sustainable. Instead, we need a strategy to achieve good performance in as few iterations as possible.

We summarize the steps of BO as follows: (i) choose and evaluate some initial points to initialize a surrogate model, usually a Gaussian Process (GP); (ii) compute the acquisition function (AF) using the surrogate model; (iii) evaluate the OF at the point that maximizes the AF; (iv) update the surrogate model. Steps (ii)-(iv) are repeated until the maximum number of iterations is reached.

We now overview the definitions of GP and AF. The *Gaussian Process* is a statistical model to describe knowledge about the OF [5]. It assumes that each unknown value of $f(x)$ originates from a Gaussian distribution whose mean $\mu_n(x)$ and variance $\sigma^2_n(x)$ depend on x and on the history of n previously observed points. Specifically, given history $H_n = \{(x_i, f(x_i)), i = 1, \ldots, n\}$ and configuration x, the distribution on $f(x)$ is the following:

$$f(x)|H_n \sim \pi_x(\cdot|H_n) = N(\mu_n(x), \sigma^2_n(x, x)). \tag{2}$$

The functions $\mu_0(\cdot)$ and $\sigma^2_0(\cdot, \cdot)$, called mean and kernel functions, respectively, are the hyperparameters of the GP model. These functions are iteratively updated after each evaluation of the OF, obtaining the *posterior distribution*, which is parametrized by $\mu_n(\cdot)$ and $\sigma^2_n(\cdot)$ as shown in Equation (2). While a constant mean function $\mu_0(\cdot) = \mu_0$ is frequently assumed, the selection of the kernel varies considerably in the literature, as it influences the smoothness of the GP model. The most popular options are the *Radial Basis Function* (RBF) and the *Matérn* covariance function [2]. Another crucial element of the BO algorithm is the choice of the *Acquisition Function* $\alpha(x)$. This function assesses the utility of evaluating $f(x)$ at a particular point x, striking a balance between exploration and exploitation. The AF is computed as a function of the *posterior distribution* on the OF and is much faster to compute than the OF itself. Consequently, at each iteration, we compute the point that maximizes the AF and then evaluate the OF at that point [2]. In this paper, we focus on the *Expected Improvement*: $\text{EI}_n(x) \coloneqq E_n\left[[f(x) - f^*_n]^+\right]$, where $f^*_n = \min_{m \leq n} f(x_m)$, i.e., the minimum value found so far. Other commonly used AFs are the Upper Confidence Bound (UCB) and the Probability of Improvement (PoI) [2].

Classical BO does not take black-box constraints into account. This issue has been addressed in MALIBOO [3], which leverages the prediction capabilities of an ML model to estimate the feasibility region directly in the AF.

B. The MALIBOO algorithm

MALIBOO is a framework implementing novel BO-based techniques for black-box functions [3]. Its main contribution consists of the integration of ML techniques within the BO algorithm to estimate whether the constraint is met for a given point. MALIBOO proposes the following ML-integrated AF:

$$\bar{\alpha}(x) = \alpha(x)\ \mathbb{1}\{\tilde{g}(x) \in [G_{min}, G_{max}]\}, \tag{3}$$

where $\tilde{g}$ is the surrogate ML model to estimate the black-box constraint function $g(x)$. This approach can apply to any AF $\alpha(x)$, like EI, UCB, and PoI. This simple approach proves effective in avoiding the evaluation of unfeasible points, which represent a waste of resources. The ML model is updated at each iteration and becomes more and more precise as the number of evaluations increases. MALIBOO also implements a memory queue mechanism for discrete features that prevents the algorithm from visiting the same point before q iterations.

In this paper, we present a framework specifically tailored for managing pure discrete variables by building upon the strengths of MALIBOO and addressing its weaknesses. Firstly, due to the limited accuracy of ML models in the initial

iterations, using an indicator function (see Equation (3)) in the AF can reduce the algorithm efficiency since it may exclude optimal points on the edge of the domain, as is often the case. Moreover, we have observed that the search process becomes stuck after a few iterations in particularly complex optimization scenarios. To address both these challenges, in the next section, we introduce novel appropriate methodologies.

IV. d-MALIBOO

Moving from continuous to discrete domains poses a challenge for BO. Consequently, even the original MALIBOO exhibits a downgrade in the performance in discrete domains, as we will show in Section V. In this section, we describe several novel approaches for effectively managing optimization scenarios involving discrete variables. We present two distinct methods for estimating the black-box constraint (Section IV-A), along with four different strategies for speeding up the search for the optimum (Section IV-B). Additionally, we incorporate the ε-greedy approach to encourage exploration when improvements appear to stall (Section IV-C). Note that all these functionalities can be used together, and henceforth, we will refer to a set of these approaches as a *combination* for *d-MALIBOO*. Finally, we summarize the complete *d-MALIBOO* algorithm in Section IV-D.

A. Evaluation of black-box constraints ("ML constraint")

Let $\alpha(x)$ be one of the original AFs for the BO algorithm. The black-box constraint $g(x) \in [G_{min}, G_{max}]$ can be estimated in two different ways:

- *Indicator*: the approach proposed in MALIBOO [3] as shown in Equation 3, where $\tilde{g}$ is a Ridge Regression model that estimates whether x lies within the constraint.
- *Probability*: train a Ridge Classifier model $\tilde{g}$ to predict the probability that the new point meets the constraint. The AF becomes:

$$\bar{\alpha}(x) = \alpha(x)\ \mathbb{P}\left(\tilde{g}(x) \in [G_{min}, G_{max}]\right). \tag{4}$$

When only a few points have been evaluated in high-dimensional domains, the ML model may lack accuracy and exclude optimal points that lie on the edge of the feasible domain. Therefore, we propose a smoother approach to integrate the evaluation of the constraint function within the AF.

B. Evaluation of the objective function ("ML target")

Dealing with vast discrete domains requires multiple evaluations of the OF to build an accurate surrogate model. We propose four different approaches that utilize the capabilities of ML to learn from collected data and predict the values of black-box functions, helping the AF in finding the optimum:

- *Indicator*: train a Ridge Regression model $\tilde{f}$ to estimate whether the OF evaluation at a new point is greater than or equal to the evaluation made at the best point thus far, i.e., f^*. The AF becomes:

$$\bar{\alpha}(x) = \alpha(x)\ \mathbb{1}\{\tilde{f}(x) \geq f^*\}. \tag{5}$$

- *Probability*: train a Ridge Classifier model $\tilde{f}$ to predict the probability that the OF evaluation at a new point is greater than or equal to the evaluation made at the best point thus far. The AF becomes:

$$\bar{\alpha}(x) = \alpha(x)\ \mathbb{P}\left(\tilde{f}(x) \geq f^*\right). \tag{6}$$

- *Sum*: train a Ridge Regression model $\tilde{f}$ and weigh the evaluation of the AF with the one of the ML model. Let $\alpha(D)$ and $\tilde{f}(D)$ be the vector of evaluations of the AF and the regression model, respectively, over the entire discrete domain D, and x the ith element in D. The AF can be rewritten as:

$$\bar{\alpha}_t(x) = (1-\gamma_t)\ m_j(\alpha(D)) + \gamma_t\ m_j(\tilde{f}(D)). \tag{7}$$

  Here, $\gamma_t \in [0, 0.5]$ is a weight parameter that increases exponentially with BO iterations t, as we expect the ML model accuracy to increase over iterations. $m_j(\cdot)$ is the min-max operator applied to the jth element[1], to scale two quantities that may have different magnitudes.
- *Product*: train a Ridge Regression model $\tilde{f}$ and scale the evaluation of the AF using the estimate from the ML model. The AF becomes:

$$\bar{\alpha}(x) = \alpha(x)\ m_j(\tilde{f}(D)). \tag{8}$$

Note that the *Sum* and *Product* techniques are highly effective but can only be applied in scenarios involving purely discrete variables, as they require the exhaustive evaluation of the AF on the discrete domain. However, the evaluation time required is remarkably small, even for large optimization domains.

C. ε-greedy approach

In our preliminary experiments, observed that in high-dimensional domains with strict constraints, the improvement in the OF evaluation may stop over the iterations. Inspired by the Reinforcement Learning literature [6], we have integrated the ε-greedy approach within the BO algorithm to address this issue. This approach involves selecting a random point with a probability of ε instead of maximizing the AF. We only choose among points that the ML model $\tilde{g}$ deems feasible.

D. The algorithm

Algorithm 1 outlines our *d-MALIBOO* technique designed specifically for handling discrete variables. At each iteration, we sample from a Bernoulli distribution with parameter ε, to establish if the next point will be selected randomly (lines 6-8). Otherwise, we update the ML models on the constraint and the OF, and consequently the AF (lines 10-11).

[1]Given a vector v, the min-max normalization operator scales its jth element as follows: $m_j(v) = \frac{v_j - \min_p v_p}{\max_p v_p - \min_p v_p} \in [0, 1]$.

Algorithm 1 *d-MALIBOO algorithm*

Input: n_0, N, GP, *ML constr.*, *ML target*, ε
Initialization: History $H \leftarrow []$
evaluate $f(\cdot)$ in n_0 initial points randomly chosen
$H \leftarrow \{(x_i, f(x_i)), i = 1, \ldots, n_0\}$
for iterations $n = 1 : N$ **do**
    ε-greedy_eval $\leftarrow$ sample from a $Be(\varepsilon)$
    if ε-greedy_eval **then**
        pick a random point x_{n+1} within *ML constr.*
    else
        train *ML constr.*, *ML target* models with data in H
        update the posterior distribution of the GP model and $\tilde{\alpha}(\cdot)$ with data in H
        find point x_{n+1} which maximizes the AF $\tilde{\alpha}(\cdot)$
    end if
    evaluate $f(x_{n+1})$, add the evaluation to H
end for
return $\widehat{x} = \arg\min_{x \in H} f(x)$

V. Experimental Results

This section shows the experimental results for validating the *d-MALIBOO* framework. In particular, in Section V-A, we describe the reference applications on which we apply the algorithms, while in Section V-B, we report the algorithms we compare with, in addition to the original MALIBOO. In Section V-C we show the experimental setup in terms of hyperparameters and hardware, and Section V-D shows the empirical results of our experimental campaign. The results of the experiments are available on Zenodo[2].

A. Reference Applications

This section presents the three applications on which we test *d-MALIBOO*. They are heterogeneous in terms of complexity, optimization domain, and, consequently, number of possible configurations. All applications are optimized over a purely discrete multidimensional domain.

1) Recipe Transcriber: The first application processes an input video to produce a transcription of the audio and identify the ingredients of a recipe by detecting them in video frames. It was developed within the OSCAR [7] framework and is deployed in the Edge-Cloud continuum. The application includes five compute-intense components deployed on AWS EC2 virtual machines[3]. We run the application within OSCAR-P [8], an automated tool for testing, deploying, and profiling containerized applications. For each component, we vary the *Number of cores* from the minimum of a single node to the maximum value that avoids resource saturation. The collection of these five values represents the input vector x. Each execution of the application involves processing a batch of 100 input videos, each 10 seconds long. The goal is to find the configuration that incurs the minimum cloud *cost* while satisfying a specified *execution time* constraint T_{max}. We consider $T_{max} \in \{65, 150, 300, 450, 600\}s$, which are significant values in the empirical distribution of times according to our preliminary analysis. We repeat the optimization experiments for each T_{max}. In the optimization process, we consider 9 randomly selected initial points and perform 30 algorithm iterations.

2) Stereomatch: The second application is an image-processing edge computing application [9] that computes the disparity value between a pair of stereo images captured from the same scene by two different cameras. This value is used to determine the depth of objects within the scene. In this case, we optimize four parameters: number of parallel cores, color similarity confidence, granularity of disparity hypotheses to be tested, and length of the support window arm. This domain is much larger than the previous application, but similar to that, we seek to minimize the *cost* while meeting an *execution time* constraint. Here, the algorithm randomly selects 3 initial points and performs 60 iterations. We set $T_{max} \in \{6, 8, 10, 12, 17, 40\}s$ based on the distribution of execution times.

3) LiGen: The third and most complex scenario involves a molecular docking application integrated into the EXSCALATE drug discovery platform [10]. The *LiGen* code simulates diverse ligand-pocket interactions, identifying promising docking poses of the ligand within the pocket through multiple restarts of a gradient descent algorithm after a clustering analysis. Then, it chooses several representative poses and evaluates them using a scoring function [11]. The quality of the docking solution is assessed by calculating the average Root Mean Square Distance (RMSD) for 100 ligand-protein pairs with a known optimal crystal position. In this scenario, a configuration x is defined by eight discrete parameters. Six of them, which control gradient descent restarts, clustering distance, and the number of poses to evaluate, impact the accuracy of the docking algorithm. The other two parameters, namely the number of CUDA threads in each block and the reading buffer size, only influence the performance of the algorithm. Fine-tuning these parameters is essential to strike the right balance between performance and accuracy. However, with the parameter space boasting over 60 million configurations, precision in tuning becomes imperative. The output features of this application are the RMSD $R(x)$ and the execution time $T(x)$. The OF to minimize is $R^3(x)T(x)$, while ensuring a quality constraint $R(x) \leq R_{max}$. Within the optimization process, we use 11 initial points and conduct 60 iterations of the optimization algorithm. The thresholds we consider are $R_{max} \in \{1.9, 2.0, 2.1, 2.2, 2.45, 2.75\}$.

B. Alternative Literature Approaches

In this section, we present the optimization algorithms from the literature that we use for comparison: *SVM-CBO* and *OpenTuner*, in addition to the original MALIBOO.

The first competing approach is Support Vector Machine Constrained BO (SVM-CBO) [12]. The SVM-CBO algorithm operates in two phases. Initially, it estimates the feasible region

[2]https://doi.org/10.5281/zenodo.11502575
[3]https://aws.amazon.com/it/ec2

within constraints using M function evaluations. Feasibility is determined by a non-linear separation hyperplane from an SVM classifier trained on these points. The next evaluation point is chosen to enhance the feasible region estimate and identify disconnected feasible regions. The second phase employs a modified BO process within the identified feasible region, fitting a GP on the OF. This approach is applied in the last $N - M$ iterations, where N is the total number of OF evaluations. To ensure a fair comparison, we use the same proportions between M and N recommended by the authors in [12]. The SVM-CBO approach is a competitor to *d-MALIBOO* since it also leverages the strengths of an ML model, i.e., SVM, to estimate the feasible region. In contrast to this approach, *d-MALIBOO* concurrently assesses both the constraints and the OF, even considering points beyond the feasible region for GP model training.

Our second competitor is *OpenTuner*, a framework designed for the development of domain-specific program autotuners [13]. One of the main strengths of this approach is the use of ensembles of search techniques. The algorithm allocates a larger testing budget to the techniques that find better configurations, while less effective techniques are assigned fewer tests or disabled outright. Techniques can share results through a shared database, collaborating to enhance the search for an optimal configuration. This popular approach is quite different from ours, but we use it as a competitor because it is a popular state-of-the-art method for constrained optimization scenarios, particularly in the HPC community.

C. Experimental Setup

In this section, we present the setup for conducting our experimental campaign and the hyperparameters chosen for *d-MALIBOO*. We execute our algorithm and the competitor methods on a Linux server featuring a 40-core Intel(R) Xeon(R) CPU, operating at 2.20 GHz, and equipped with 32 GB of memory. The total algorithm execution time for *d-MALIBOO* to select configurations does not exceed 23 minutes in the worst case. To ensure the robustness of the experimental analysis, we conduct each experiment $K = 30$ times with different random seeds, resulting in a varied choice of initial points. We then evaluate the performance of the algorithms by computing the following metrics:

- *Mean Absolute Percentage Regret* (MAPR): the relative difference between the best solution found and the real optimum, averaged over the repetitions:

$$MAPR(\hat{\mathbf{f}}, f^*) = \frac{1}{K}\sum_{k=1}^{K}\left|\frac{\hat{f}^k - f^*}{f^*}\right| * 100\%, \quad (9)$$

  where $\hat{\mathbf{f}}$ is the vector of best OF values for each repetition, and f^* is the ground-truth minimum of the OF.
- *percentage standard deviation*: the standard deviation of the optimal values over the K repetitions, normalized on the real optimum. This normalization enables a fair comparison across different values of f^*:

$$\%\ std\ dev = \frac{\sigma}{f^*} * 100\%. \quad (10)$$

  Note that in our experiments, some repetitions did not find any feasible configuration for certain algorithms. In the computation of the MAPR and the standard deviation, we only include repetitions that found at least one.
- *percentage of feasibility*: the percentage of repetitions that return a feasible solution, i.e., that find at least one feasible point during the optimization process.

As recommended in [3], we adopt a constant mean function $\mu(\cdot) = \mu_0$ and a Matérn kernel for both *d-MALIBOO* and MALIBOO. When using the ε-greedy approach, we fix $\varepsilon = 0.1$. Additionally, for both the constraint and target function prediction ML models, we employ Ridge regression models with 2nd-degree polynomial feature expansion, which yield fast and accurate predictions. Based on our preliminary analysis, the test-set Mean Absolute Percentage Error for this model is nearly always below 9% [3].

D. Empirical Results

In this section, we present the empirical results to validate the proposed *d-MALIBOO* algorithms. First, we combine the features introduced in Section IV to determine the best *d-MALIBOO* combination in terms of MAPR and percentage of feasibility. Then, we compare our results with those obtained using SVM-CBO and OpenTuner. Finally, we analyze the benefits of *d-MALIBOO* based on the results.

1) Best d-MALIBOO combination: We test all 20 possible combinations of the *d-MALIBOO* algorithm to determine the most effective in terms of regret and feasibility of configurations. As an example, Table I shows the results obtained with *Stereomatch*, fixing $T_{max} = 12$s. It ranks combinations according to the following metric:

$$\frac{MAPR\ +\ 2 * stddev}{feasibility}. \quad (11)$$

ML constr.	ML target	ε-greedy	MAPE	stdev [%]	feas. [%]
probability	product	False	8.10	12.70	96.67
indicator	product	False	9.67	12.05	96.67
probability	probability	True	8.31	14.97	100.00
probability	sum	False	12.44	14.12	100.00
probability	probability	False	13.06	13.87	100.00
probability	None	False	14.48	13.23	100.00
indicator	probability	False	10.56	17.18	100.00
indicator	sum	True	11.46	16.88	100.00
indicator	sum	False	10.50	17.73	100.00
probability	sum	True	16.52	15.96	100.00
indicator	probability	True	12.36	19.56	100.00
probability	None	True	15.10	24.08	100.00
probability	product	True	14.95	25.57	100.00
probability	indicator	False	17.71	51.63	100.00
probability	indicator	True	27.06	72.53	100.00
indicator	product	True	25.84	84.30	100.00
indicator	**None**	**False**	**35.44**	**83.23**	**100.00**
indicator	None	True	35.76	84.57	100.00
indicator	indicator	True	39.29	94.82	100.00
indicator	indicator	False	57.05	117.37	100.00

TABLE I: Results of *d-MALIBOO* on Stereomatch with $T_{max} = 12s$. We highlight MALIBOO in bold blue. We rank combinations according to the metric in Equation (11).

This metric can be interpreted as an upper confidence bound on the regret, normalized on the percentage of feasible results.

Using the standard deviation of the regret in our ranking metric, in addition to the mean, gives a greater degree of statistical robustness to our results.

Finally, we collect data from experiments with all applications and constraint bounds, compute the rankings of all *d-MALIBOO* combinations, and average these rankings to determine the best one. Table II shows the five combinations with the best average ranking.

ML constraint	ML target	ε-greedy	Label
indicator	probability	True	Config. 1
probability	probability	True	Config. 2
indicator	product	True	Config. 3
probability	probability	False	Config. 4
indicator	indicator	False	Config. 5

TABLE II: Best five *d-MALIBOO* combinations.

These results show that the features introduced in *d-MALIBOO* perform especially well in conjunction with each other. The most effective combination appears to be the one that uses the "indicator" as the estimator of the black-box constraint, the "probability" approach applied to estimate the OF and the ε-greedy approach to favour exploration.

2) Comparison with the state of the art: As a final analysis, we compare the best combinations of our *d-MALIBOO* algorithm with SVM-CBO and OpenTuner. In particular, Tables III, IV, and V show the results for *Recipe Transcriber*, *Stereomatch* and *LiGen*, respectively, with all tested constraints.

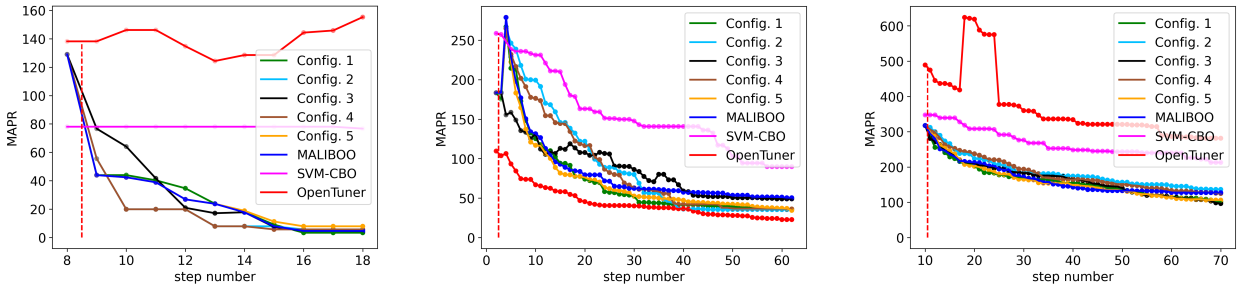
For *Recipe Transcriber*, we observe that different *d-MALIBOO* combinations achieve the best results depending on the value of the constraint. In this scenario, the original MALIBOO shows results in line with *d-MALIBOO*, likely because the optimization domain is relatively small. Additionally, both competitor approaches perform significantly worse than all *d-MALIBOO* combinations. For *Stereomatch*, OpenTuner performs best, especially in terms of feasible solutions found when the constraint is particularly tight. It is the only approach to achieve an acceptable percentage of feasibility with $T \leq 6s$. However, its performance deteriorates compared to all other approaches as the constraint becomes less restrictive. The combination labeled *Config. 5* in Table IV is generally the most effective with this application, while the original MALIBOO struggles to lower the regret, especially under strict constraints. Notice that a significant number (25 to 100%, depending on the threshold) of SVM-CBO repetitions do not find any feasible configuration. Finally, we comment the results for *LiGen*, which represents the most challenging optimization problem in this work. *Config. 3* is the combination that performs best for all thresholds, while both state-of-the-art methods perform generally worse than *d-MALIBOO*.

In Figure 1, we show the MAPR trend for specific thresholds of the three reference applications. Notice that for *Recipe Transcriber* we plot the results up to the tenth optimization iteration, as the improvement in all algorithms stalls beyond that point. We observe that the decrease in regret for SVM-CBO is slower than for *d-MALIBOO* combinations. With the Stereomatch application, on average, OpenTuner ultimately reaches a worse configuration than *d-MALIBOO*, despite achieving faster convergence in the early iterations.

Algorithm	Constraint	MAPE	stdev [%]	feas. [%]
Config. 1	$T \leq 65s$	0.07	0.38	96.67
Config. 2		**0.07**	**0.37**	**100.00**
Config. 3		0.07	0.38	96.67
Config. 4		**0.07**	**0.37**	**100.00**
Config. 5		0.07	0.38	96.67
MALIBOO		0.07	0.38	96.67
SVM-CBO		inf	-	0.00
OpenTuner		inf	-	0.00
Config. 1	$T \leq 150s$	**3.34**	**13.16**	**100.00**
Config. 2		5.68	17.54	100.00
Config. 3		4.47	14.24	100.00
Config. 4		5.68	17.54	100.00
Config. 5		7.88	20.58	100.00
MALIBOO		4.60	14.53	100.00
SVM-CBO		76.70	60.44	10.00
OpenTuner		155.25	63.71	43.33
Config. 1	$T \leq 300s$	14.55	26.35	100.00
Config. 2		23.84	34.82	100.00
Config. 3		**11.14**	**25.76**	**100.00**
Config. 4		15.89	29.77	100.00
Config. 5		23.55	33.76	100.00
MALIBOO		14.04	28.97	100.00
SVM-CBO		59.51	38.92	76.67
OpenTuner		73.73	20.32	100.00
Config. 1	$T \leq 450s$	17.92	31.31	100.00
Config. 2		15.51	29.70	100.00
Config. 3		26.51	37.63	100.00
Config. 4		**14.09**	**29.17**	**100.00**
Config. 5		31.82	37.26	100.00
MALIBOO		15.68	31.36	100.00
SVM-CBO		63.54	36.96	13.33
OpenTuner		87.60	21.53	100.00
Config. 1	$T \leq 600s$	23.20	32.33	100.00
Config. 2		**19.69**	**29.83**	**100.00**
Config. 3		26.73	36.90	100.00
Config. 4		23.52	33.68	100.00
Config. 5		22.74	33.33	100.00
MALIBOO		17.56	31.30	100.00
SVM-CBO		inf	-	0.00
OpenTuner		87.96	13.57	100.00

TABLE III: Results of the five best combinations of *d-MALIBOO*. SVM-CBO and OpenTuner on the *Recipe Transcriber* application. In **bold** the best result for each constraint.

3) Discussion: In general, these results show that the use of a "ML target" approach is the most effective, especially for complex applications. The approach used by the original MALIBOO for assessing the feasibility region, i.e., the "indicator", is effective when combined with the ML models for the OF. Furthermore, as expected, the ε-greedy approach is most effective for cases with a large number of features, in which it is easier for improvements to stall due to the curse of dimensionality. We also observe that *d-MALIBOO* outperforms the two state-of-the-art methods in almost any application and for any value of the threshold. OpenTuner is more effective only when the constraint significantly restricts the feasibility domain for *Stereomatch*. Conversely, SVM-CBO is rarely competitive with either *d-MALIBOO* or OpenTuner. A possible explanation may be that for extreme values of the constraint, i.e., when the feasibility region is either very small compared to the domain or almost as large as the latter, the SVM-CBO algorithm is unable to define the boundary of the feasibility



(a) *Recipe Transcriber*, $T(x) \leq 150s$ (b) *Stereomatch*, $T(x) \leq 8s$ (c) *LiGen*, $R(x) \leq 2.45$

Fig. 1: MAPR trend for some particular runs.

Algorithm	Constraint	MAPE	stdev [%]	feas. [%]
Config. 1	$T \leq 6s$	19.89	7.44	6.67
Config. 2		62.60	44.27	6.67
Config. 3		63.30	47.02	16.67
Config. 4		23.98	5.65	6.67
Config. 5		27.18	22.48	6.67
MALIBOO		50.03	37.01	6.67
SVM-CBO		91.58	72.33	13.33
OpenTuner		**28.66**	**47.27**	**96.67**
Config. 1	$T \leq 8s$	35.64	17.49	100.00
Config. 2		35.38	27.26	100.00
Config. 3		49.00	41.81	100.00
Config. 4		36.16	16.90	100.00
Config. 5		34.43	13.68	100.00
MALIBOO		50.41	40.17	100.00
SVM-CBO		89.82	51.17	60.00
OpenTuner		**22.76**	**16.80**	**100.00**
Config. 1	$T \leq 10s$	22.58	22.46	100.00
Config. 2		17.50	14.37	100.00
Config. 3		24.41	30.11	100.00
Config. 4		15.04	12.75	100.00
Config. 5		**12.04**	**13.30**	**100.00**
MALIBOO		16.54	15.55	100.00
SVM-CBO		50.10	29.88	93.33
OpenTuner		21.63	13.47	100.00
Config. 1	$T \leq 12s$	12.36	19.56	100.00
Config. 2		**8.31**	**14.97**	**100.00**
Config. 3		25.84	84.30	100.00
Config. 4		13.06	13.87	100.00
Config. 5		57.05	117.37	100.00
MALIBOO		35.44	83.23	100.00
SVM-CBO		59.50	46.23	80.00
OpenTuner		27.77	17.44	100.00
Config. 1	$T \leq 17s$	5.91	14.44	100.00
Config. 2		3.09	4.47	100.00
Config. 3		10.08	13.79	100.00
Config. 4		7.42	9.92	100.00
Config. 5		**1.87**	**4.79**	**100.00**
MALIBOO		15.70	41.52	100.00
SVM-CBO		40.73	41.64	50.00
OpenTuner		16.27	21.17	100.00
Config. 1	$T \leq 40s$	0.90	1.56	100.00
Config. 2		0.37	0.63	100.00
Config. 3		0.74	1.37	100.00
Config. 4		0.30	0.64	100.00
Config. 5		**0.02**	**0.07**	**100.00**
MALIBOO		7.60	38.77	100.00
SVM-CBO		0.39	0.55	10.00
OpenTuner		12.37	10.80	100.00

TABLE IV: Results of the five best combinations of *d-MALIBOO*. SVM-CBO and OpenTuner on the *Stereomatch* application. In **bold** the best result for each constraint.

Algorithm	Constraint	MAPE	stdev [%]	feas. [%]
Config. 1	$R \leq 1.9$	122.40	103.04	66.67
Config. 2		133.47	47.83	16.67
Config. 3		**103.70**	**62.55**	**90.00**
Config. 4		150.90	74.35	16.67
Config. 5		126.14	123.72	66.67
MALIBOO		139.99	119.89	66.67
SVM-CBO		217.57	96.86	33.33
OpenTuner		212.22	183.25	86.67
Config. 1	$R \leq 2.0$	171.02	118.44	96.67
Config. 2		178.75	67.80	70.00
Config. 3		**167.41**	**56.73**	**100.00**
Config. 4		234.48	215.77	70.00
Config. 5		206.06	162.72	96.67
MALIBOO		211.06	161.80	96.67
SVM-CBO		244.26	89.02	83.33
OpenTuner		285.60	138.70	100.00
Config. 1	$R \leq 2.1$	199.09	334.79	100.00
Config. 2		144.31	42.37	93.33
Config. 3		**125.32**	**37.14**	**100.00**
Config. 4		144.62	53.65	93.33
Config. 5		180.01	202.55	100.00
MALIBOO		144.71	72.86	100.00
SVM-CBO		195.79	56.11	100.00
OpenTuner		221.77	92.79	100.00
Config. 1	$R \leq 2.2$	151.32	40.79	100.00
Config. 2		156.88	45.02	96.67
Config. 3		**146.31**	**36.38**	**100.00**
Config. 4		169.47	38.47	96.67
Config. 5		133.30	46.70	100.00
MALIBOO		153.72	44.04	100.00
SVM-CBO		222.87	55.65	100.00
OpenTuner		292.25	123.03	100.00
Config. 1	$R \leq 2.45$	101.09	41.31	100.00
Config. 2		136.42	37.70	100.00
Config. 3		**96.32**	**41.34**	**100.00**
Config. 4		124.78	39.97	100.00
Config. 5		106.16	47.37	100.00
MALIBOO		127.28	35.27	100.00
SVM-CBO		212.82	58.83	60.00
OpenTuner		281.49	120.58	100.00
Config. 1	$R \leq 2.75$	144.48	53.69	100.00
Config. 2		167.58	49.36	100.00
Config. 3		**134.20**	**52.90**	**100.00**
Config. 4		175.30	51.53	100.00
Config. 5		150.49	47.19	100.00
MALIBOO		189.56	88.63	100.00
SVM-CBO		264.35	97.75	6.67
OpenTuner		348.14	106.97	100.00

TABLE V: Results of the five best combinations of *d-MALIBOO*. SVM-CBO and OpenTuner on the *LiGen* application. In **bold** the best result for each constraint.

region during phase 1. If the points explored in phase 1 are all feasible or unfeasible, the algorithm stops due to the lack of an estimated boundary. Ultimately, *d-MALIBOO* outperforms state-of-the-art approaches, reducing regret by an average of 2 to 8 times. Additionally, in some cases such as *Recipe Transcriber* with $T \leq 150s$ and *Stereomatch* with $T \leq 40s$, the improvement is in the tens of times.

VI. Related Work

In recent years, using BO with discrete variables has gained popularity as a means of conducting hyperparameter tuning of ML algorithms. In [4], the authors propose straightforward solutions for dealing with integer-valued and categorical variables, such as rounding for integer-valued variables and one-hot encoding for categorical ones. These solutions lead to a GP model that does not consider the fact that the OF is constant in regions approximated to the same integer values, making it an inaccurate model for the OF. To address this problem, the authors introduce a variable transformation, which yields an alternative GP covariance function that accurately models those constant regions, thereby improving the outcome of the optimization process. Another work [14] introduces MIVABO, a BO algorithm designed for optimizing mixed-variable functions subject to known linear and quadratic integer constraints. MIVABO separates continuous, discrete, and mixed components of the OF using a linear surrogate model. In this manner, the algorithm efficiently draws samples from the posterior distribution using Thompson sampling to optimize the constrained AF. In [15], the authors propose Discrete-BO to solve the problem of repeating previously observed observations. They reparametrize the parameters of the UCB AF and the RBF kernel of the GP based on whether the suggested point has been already visited. The Probabilistic Reparameterization (PR) approach presented in [16] maximizes the expectation of the AF over a probability distribution defined over continuous parameters rather than directly optimizing the AF over discrete parameters. This is made possible through appropriate reparameterizations of discrete variables into continuous ones. Finally, [17] introduces CoCaBO, an approach that combines the benefits of multi-armed bandits and BO to manage both continuous and categorical inputs. It uses a kernel designed for mixed-type spaces, which allows capturing the interactions between continuous and categorical inputs without needing one-hot transformations.

VII. Conclusions and Future Work

In this work, we have presented *d-MALIBOO*, a collection of algorithms that integrates BO and ML to address the optimization of constrained problems in discrete and multi-dimensional domains. Specifically, we have developed novel methods for evaluating black-box constraints more smoothly than [3] and for enhancing the search for the optimum in discrete domains. Additionally, we have implemented an ε-greedy approach to prevent stagnation in the improvement of percentage regret (MAPR). The framework allows for any combination of ML models on the bounds, target, and with the ε-greedy approach. We have tested *d-MALIBOO* combinations, highlighting the best ones. Finally, we have compared *d-MALIBOO* with MALIBOO, SVM-CBO, and OpenTuner, observing an improvement of 2 to 8 times in terms of MAPR.

Future work will focus on exploiting adaptive methods to further improve the *d-MALIBOO* algorithm, identifying the optimal combination of parameters, and developing a tool to tackle mixed-variable optimization problems.

References

[1] B. Jennings and R. Stadler, "Resource management in clouds: Survey and research challenges," *Journal of Network and Systems Management*, vol. 23, pp. 567–619, 2015.

[2] P. I. Frazier, "A tutorial on Bayesian optimization," *arXiv preprint arXiv:1807.02811*, 2018.

[3] B. Guindani, D. Ardagna, A. Guglielmi, R. Rocco, and G. Palermo, "Integrating Bayesian optimization and machine learning for the optimal configuration of cloud systems," *IEEE Transactions on Cloud Computing*, vol. 12, no. 1, pp. 277–294, 2024.

[4] E. C. Garrido-Merchán and D. Hernández-Lobato, "Dealing with categorical and integer-valued variables in Bayesian optimization with Gaussian processes," *Neurocomputing*, vol. 380, pp. 20–35, 2020.

[5] C. E. Rasmussen and C. K. Williams, *Gaussian processes for machine learning*. Springer, 2006, vol. 1.

[6] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[7] S. Risco, G. Moltó, D. M. Naranjo, and I. Blanquer, "Serverless workflows for containerised applications in the cloud continuum," *Journal of Grid Computing*, vol. 19, pp. 1–18, 2021.

[8] E. Galimberti, B. Guindani, F. Filippini, H. Sedghani, D. Ardagna, G. Moltó, and M. Caballer, "Oscar-p and amllibrary: Performance profiling and prediction of computing continua applications," in *Companion of the 2023 ACM/SPEC ICPE*, 2023.

[9] E. Paone, G. Palermo, V. Zaccaria, C. Silvano, D. Melpignano, G. Haugou, and T. Lepley, "An exploration methodology for a customizable opencl stereo-matching application targeted to an industrial multi-cluster architecture," in *CODES+ISSS*, 2012.

[10] D. Gadioli, E. Vitali, F. Ficarelli, C. Latini, C. Manelfi, C. Talarico, C. Silvano, C. Cavazzoni, G. Palermo, and A. R. Beccari, "Exscalate: an extreme-scale virtual screening platform for drug discovery targeting polypharmacology to fight sars-cov-2," *IEEE Transactions on Emerging Topics in Computing*, vol. 11, no. 1, pp. 170–181, 2022.

[11] E. Vitali, D. Gadioli, G. Palermo, A. Beccari, C. Cavazzoni, and C. Silvano, "Exploiting openmp and openacc to accelerate a geometric approach to molecular docking in heterogeneous hpc nodes," *The Journal of Supercomputing*, vol. 75, pp. 3374–3396, 2019.

[12] A. Candelieri, "Sequential model based optimization of partially defined functions under unknown constraints," *Journal of Global Optimization*, vol. 79, no. 2, pp. 281–303, 2021.

[13] J. Ansel, S. Kamil, K. Veeramachaneni, J. Ragan-Kelley, J. Bosboom, U.-M. O'Reilly, and S. Amarasinghe, "Opentuner: An extensible framework for program autotuning," in *IEEE PACT*, 2014, pp. 303–316.

[14] E. Daxberger, A. Makarova, M. Turchetta, and A. Krause, "Mixed-variable Bayesian optimization," *arXiv preprint arXiv:1907.01329*, 2019.

[15] P. Luong, S. Gupta, D. Nguyen, S. Rana, and S. Venkatesh, "Bayesian optimization with discrete variables," in *AJCAI*. Springer, 2019, pp. 473–484.

[16] S. Daulton, X. Wan, D. Eriksson, M. Balandat, M. Osborne, and E. Bakshy, "Bayesian optimization over discrete and mixed spaces via probabilistic reparameterization," *NeurIPS*, 2022.

[17] B. Ru, A. Alvi, V. Nguyen, M. A. Osborne, and S. Roberts, "Bayesian optimisation over multiple continuous and categorical inputs," in *ICML*, 2020.