Automating Pareto-Optimal Experiment Design via Efficient Bayesian Optimization

by

Yunsheng Tian

B.Eng., Nankai University (2019)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2021

Author .....
Department of Electrical Engineering and Computer Science
May 7, 2021

Certified by .....
Wojciech Matusik
Professor of Electrical Engineering and Computer Science
Thesis Supervisor

Accepted by .....
Leslie A. Kolodziejski
Professor of Electrical Engineering and Computer Science
Chair, Department Committee on Graduate Students

Automating Pareto-Optimal Experiment Design via Efficient Bayesian Optimization

by

Yunsheng Tian

Submitted to the Department of Electrical Engineering and Computer Science
on May 7, 2021, in partial fulfillment of the
requirements for the degree of
Master of Science

Abstract

Many science, engineering, and design optimization problems require balancing the trade-offs between several conflicting objectives. The objectives are often blackbox functions whose evaluation requires time-consuming and costly experiments. Multi-objective Bayesian optimization can be used to automate the process of discovering the set of optimal solutions, called Pareto-optimal, while minimizing the number of performed evaluations. To further reduce the evaluation time in the optimization process, testing of several samples in parallel can be deployed. We propose DGEMO, a novel multi-objective Bayesian optimization algorithm that iteratively selects the best batch of samples to be evaluated in parallel. Our algorithm approximates and analyzes a piecewise-continuous Pareto set representation, which allows us to introduce a batch selection strategy that optimizes for both hypervolume improvement and diversity of selected samples in order to efficiently advance promising regions of the Pareto front. Experiments on both synthetic test functions and real-world benchmark problems show that our algorithm predominantly outperforms relevant state-of-the-art methods. The code is available at `https://github.com/yunshengtian/DGEMO`.

In addition, we present AutoOED, an Optimal Experiment Design platform that implements several multi-objective Bayesian optimization algorithms with state-of-the-art performance including DGEMO with an intuitive graphical user interface (GUI). AutoOED is open-source and written in Python. The codebase is modular, facilitating extensions and tailoring the code, serving as a testbed for machine learning researchers to easily develop and evaluate their own multi-objective Bayesian optimization algorithms. Furthermore, a distributed system is integrated to enable parallelized experimental evaluations by independent workers in remote locations. The platform is available at `https://autooed.org`.

Thesis Supervisor: Wojciech Matusik
Title: Professor of Electrical Engineering and Computer Science

Acknowledgments

First of all, I would like to thank my thesis supervisor Prof. Wojciech Matusik, for his consistent support, insightful ideas, and encouragement which helped me complete 1/3 (hopefully) of my PhD journey. Under his guidance, I felt both my taste and methodologies of academic research are much improved. I hope the next several years would be more rewarding, and more impactful works will appear in my PhD thesis.

Next, I would like to thank my collaborators who contributed to important parts of this thesis, including Mina Konaković Luković, Timothy Erps, and Michael Foshey. I had an enjoyable time collaborating with these folks, and they offered great help to these projects to make my thesis happen. This thesis is supported by National Science Foundation. And I would like to acknowledge support and feedback for this thesis from Konica Minolta.

I would also like to thank all my past collaborators, especially Jie Xu, Pingchuan Ma, Prof. Daniela Rus, and Prof. Shinjiro Sueda, for their significant help on my first publication since I started as a graduate student at MIT. And I am honored to spend a pleasant two years with my talented labmates in Computational Design & Fabrication Group, even though most of the time on Zoom, unfortunately. I am looking forward to our reunion in person and the barbeque by the Charles River.

MIT is a wonderful place with not only hardworking peers but also interesting and supportive friends. I want to thank you all for the great time we had, the game we played, the tequila we drank, the chat we had about research and the future.

I want to thank Boston because of my fiancee Yue Liu. We met here in spring, we chased the sunset in summer, we had our own home in fall, and in winter, we knew we would accompany each other throughout our life. Thank you for giving me all the best memories and hopes. Thank you for making my life brighter than ever.

Finally, I would like to thank my family for their unconditional trust, understanding, and support, which makes me who I am today. I know we are always emotionally connected, even though physically separated by the ocean. I hope this thesis could bring me one step closer to repaying your expectation and love.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Optimal Experiment Design (OED) problems in science and engineering often require satisfying several conflicting objectives simultaneously. These problems aim to solve a multi-objective optimization system and discover a set of optimal solutions, called Pareto optimal. Furthermore, the objectives are typically black-box functions whose evaluations are time-consuming and costly (e.g., evaluations can either measure real experiments or run expensive numerical simulations). Thus, the budget that determines the number of experiments can be heavily constrained. Hence, an efficient strategy for guiding the experimental design towards Pareto-optimal solutions is necessary.

Recent advances in machine learning algorithms have facilitated optimization of various design problems, including chemical design [22], material design [59], battery design [2], clinical drug trials [56], resource allocation [55], environmental monitoring [35], robotics [36]. An approach that has proven to be powerful in optimizing the expensive-to-evaluate black-box functions and enables automatic guidance of the design process is Bayesian optimization (BO) [26, 48]. A fundamental concept behind the BO is two-fold: first, a inexpensive surrogate model is used to model the black-box objective function based on the evaluated experiments; second, an acquisition function is employed to adaptively sample the design space and efficiently improve the set of optimal solutions. Both single- and multi-objective Bayesian optimization (MOBO) have been studied.

In a variety of physical experimental setups, evaluating several experiments in

parallel, e.g., in batches, reduces the time and cost of the optimization process. Experiments can take days, weeks, even months to complete; hence taking advantage of batching is important. Furthermore, in design problems with multiple conflicting objectives, in most cases, there is no single optimal solution, but rather a set of optimal designs with different trade-offs. These designs often tend to group in different regions based on their properties and performance. In this thesis, we develop a novel MOBO approach with a batch selection strategy that combines the knowledge from both design and performance space. Our selection strategy enforces the exploration of diverse identified regions in approximated optimal space. In addition, it measures which regions are the most beneficial for optimization to explore by taking into consideration the *hypervolume improvement*[45]. A relatively limited amount of literature can be found on the batch selection for MOBO that applies to the problem setup we are trying to address [8, 58]. To the best of our knowledge, no previous published work has considered diversity in both design and performance space in the batch selection strategy.

Even though the concept of BO is extensively studied in the machine learning and optimization community from a theoretical aspect and in the single-objective case, its potential applications in other fields with multi-objective problems are still not widely explored due to the lack of intuitive, easy to use, and open-source software. To fill this gap, we propose an open-source platform which is based on the concept of MOBO and is entirely implemented in Python in a modular way. This platform is powered by not only the state-of-the-art MOBO algorithms, but also an intuitive graphical user interface which is easy to use. Additionally, this platform offers many advanced features for practical use cases, for example, asynchronous batch evaluations, automated experiment design, and distributed collaboration. More details can be found in Chapter 6.

In summary, the key contributions of this thesis are the following:

- We propose a novel approach, referred to as DGEMO (Diversity-Guided Efficient Multi-objective Optimization), for solving multi-objective optimization problems of black-box functions, while minimizing the number of function evaluations.

Our algorithm is based on multi-objective Bayesian optimization that operates on batches of evaluated samples in each iteration and can handle an arbitrary batch size.

- Our batch selection strategy is guided by diversity in both design and performance space of selected samples. The selection strategy divides the space into diversity regions that provide additional information in terms of shared properties and performance of optimal solutions. This information can be valuable for further physical experiments and problem understanding.

- We conduct comprehensive experiments on both synthetic test functions and real-world benchmark problems. Our algorithm exhibits consistent high-quality performance and outperforms the relevant existing works in the field.

- We propose AutoOED, an open-source platform for multi-objective optimization problems with a restricted budget of test samples. The platform automatically guides the design of experiments to be evaluated and quickly leads to the best performing designs, which can be easily used by people from any industry without any background in coding and optimization.

Chapter 2

Related Work

2.1 Bayesian optimization (BO)

BO originated from [30, 39], and was popularized by the Efficient Global Optimization (EGO) algorithm [26]. BO achieves a minimal number of function evaluations by utilizing the surrogate model and sampling guided by carefully designed selection criteria. There are two types of selection criteria: *sequential selection* that proposes only a single optimal sample to evaluate, and *batch selection* proposing a set of samples to evaluate in each iteration. The sequential selection has long been investigated in the literature [30, 26, 50, 24], which usually achieves better performance w.r.t. the number of evaluations, at the cost of more algorithm iterations and slower convergence. On the other hand, batch selection sacrifices some performance, yet can benefit from a parallel evaluation, which is more common and time-efficient in modern engineering experiments [17, 11, 27, 21]. In this thesis, we focus on the time-efficient batch selection scheme.

2.2 Multi-objective optimization (MOO)

MOO is applied to problems involving several conflicting objectives and optimizes for a set of Pareto-optimal solutions [38]. To this end, many population-based multi-objective evolutionary algorithms (MOEA) are proposed, e.g., NSGA-II [14] and

MOEA/D [57], which are widely used in various multi-objective problems, including financial portfolio design [51], manufacturing design [47], and optimal control [19]. However, MOEA algorithms typically require a substantial number of evaluations due to the nature of evolutionary computation. This requirement prevents them from being applied to many real-world problems, where the evaluation can be computationally expensive and becomes the main bottleneck of the whole optimization process.

2.3 Multi-objective Bayesian optimization (MOBO)

Taking the best of both worlds from Bayesian optimization and multi-objective optimization, MOBO is designed to solve multi-objective problems that are expensive to evaluate. Among MOBO algorithms there are also two streams of work focusing on sequential selection and batch selection; we call them the *single-point* method and *batch* method. One of the first single-point methods is ParEGO [28], which randomly scalarizes the multi-objective problem to a single-objective problem and selects a sample with maximum *expected improvement* (EI). Similarly, EHI [18] and SUR [42] extend ParEGO by using different acquisition functions. Another line of work, such as PAL [62], focuses on uncertainty reduction on the surrogate model for better performance. PESMO [46] and MESMO [5] rely on entropy-based acquisition functions and select a sample that maximizes the information gain about the optimal Pareto set. A very recent approach USeMO [4] uses maximum uncertainty as a selection criterion based on the Pareto set generated by NSGA-II. USeMO achieves state-of-the-art performance and generalizes to any acquisition function. Even though these algorithms fall into the category of the single-point method, some of them could have a straight-forward extension to the batch selection scheme, which we demonstrate in Chapter 5.

For batch MOBO methods, MOEA/D-EGO [58] can be considered as the earliest attempt, which generalizes ParEGO by multiple scalarization weights and performs parallel optimization with MOEA/D. A subsequent work MOBO/D is proposed as an extension to MOEA/D-EGO by changing the acquisition function [31]. A recent

work TSEMO [8] suggests using Thompson Sampling (TS) on the GP posterior as an acquisition function, optimizing multiple objectives with NSGA-II, and selecting the next batch of samples by maximizing the hypervolume improvement. BS-MOBO [32] applies MOBO on a different setting with large scale datasets using neural network as surrogate model. However, to the best of our knowledge, none of the previous methods consider diversity in both design and performance space in the batch selection, which can be crucial for many real-world problems where the Pareto-optimal solutions are distributed in diverse regions of the design space.

2.4 Open source optimal experiment design software

The theoretical concept and algorithms of BO and MOBO have been well studied in the machine learning community, but they are not widely adopted by the industries for real-world use cases mainly because of the lack of easy-to-use, intuitive and open-source software, especially for MOBO. For example, some relevant work includes BoTorch [3], which is a recent library for Bayesian optimization, nevertheless, it lacks a GUI, does not support distributed usage, and is targeted at experts in coding. Auto-QChem [49] is a recent open-source software for experiment design in chemistry, but it does not optimize for multiple objectives and does not apply to fields other than chemistry. To the best of our knowledge, our proposed platform is the first Pareto-optimal experiment design software with state-of-the-art algorithms implemented and is a general tool that can be applied to relevant problems from almost any industry and used by people without any background knowledge in coding and optimization.

Chapter 3

Preliminaries

3.1 Multi-objective optimization

We consider a multi-objective optimization problem over a set of continuous input variables $\mathcal{X} \subset \mathbb{R}^d$, called *design space*. The goal is to simultaneously minimize $m \geq 2$ objective functions $f_1, ..., f_m : \mathcal{X} \to \mathbb{R}$. We denote the vector of all objectives as $\mathbf{f}(\mathbf{x}) = (f_1(\mathbf{x}), ..., f_m(\mathbf{x}))$, where $\mathbf{x} \in \mathcal{X}$ is a vector of input variables. The *performance space* is then an m-dimensional image $\mathbf{f}(\mathcal{X}) \subset \mathbb{R}^m$.

Objectives are often conflicting, resulting in a set of optimal solutions, rather than a single best solution. These optimal solutions are referred to as *Pareto set* $\mathcal{P}_s \subseteq \mathcal{X}$ in the design space, and the corresponding images in performance space are *Pareto front* $\mathcal{P}_f = \mathbf{f}(\mathcal{P}_s) \subset \mathbb{R}^m$. More formally, a point $\mathbf{x}^* \in \mathcal{P}_s$ is considered Pareto-optimal if there is no other point $\mathbf{x} \in \mathcal{X}$ such that $f_i(\mathbf{x}^*) \geq f_i(\mathbf{x})$ for all i and $f_i(\mathbf{x}^*) > f_i(\mathbf{x})$ for at least one i.

To measure the quality of an approximated Pareto front, *hypervolume indicator* [61] is the most commonly used metric in MOO [45]. Let $\mathcal{P}_f$ be a Pareto front approximation in an m-dimensional performance space and given a reference point $\boldsymbol{r} \in \mathbb{R}^m$, the hypervolume $\mathcal{H}(\mathcal{P}_f)$ is defined as

$$\mathcal{H}(\mathcal{P}_f) = \int_{\mathbb{R}^m} \mathbb{1}_{H(\mathcal{P}_f)}(z)dz \tag{3.1}$$

where $H(\mathcal{P}_f) = \{\boldsymbol{z} \in Z \mid \exists 1 \leq i \leq |\mathcal{P}_f| : \boldsymbol{r} \preceq \boldsymbol{z} \preceq \mathcal{P}_f(i)\}$. $\mathcal{P}_f(i)$ is the i-th solution in $\mathcal{P}_f$, $\preceq$ is the relation operator of objective dominance, and $\mathbb{1}_{H(\mathcal{P}_f)}$ is a Dirac delta function that equals 1 if $z \in H(P_f)$ and 0 otherwise. The higher the hypervolume, the better $\mathcal{P}_f$ approximates the true Pareto front. To determine how much the hypervolume would increase if a set of new points $P = \{\mathbf{p}_1, \ldots, \mathbf{p}_n\} \subset \mathbb{R}^m$ is added to the current Pareto front approximation $\mathcal{P}_f$, we can use the *hypervolume improvement* that is defined as

$$\mathrm{HVI}(P, \mathcal{P}_f) = \mathcal{H}(\mathcal{P}_f \cup P) - \mathcal{H}(\mathcal{P}_f). \tag{3.2}$$

3.2 Bayesian optimization

A variety of optimization problems search to find a global optimal solution of a black-box function $f : \mathcal{X} \subset \mathbb{R}^d \to \mathbb{R}$ that is expensive to evaluate. Neither analytical form nor derivatives of f are known, and a limited number of function evaluations are available. In this scenario, Bayesian optimization (BO) has proven to be a powerful tool. The key advantage of BO lies in the selection strategy of the next points to evaluate that balances the trade-off between exploration of unknown regions vs. exploitation of the best-performing ones. A good selection strategy allows BO to achieve great results in a small number of algorithm iterations and function evaluations. In each iteration of the BO algorithm, the first step is to train a statistical model on all available evaluated points, called *surrogate model*. Typically a Gaussian process is used to model the unknown objective function f and provide the model's prediction and uncertainty. The second step of the algorithm is the optimization of an *acquisition function* on the surrogate model that selects a point to evaluate next. The most popular acquisition functions for single-objective optimization problems include *expected improvement* (EI) [39, 26], *probability of improvement* (PI) [30], and *upper confidence bound* (UCB) [50]. In the case of multi-objective optimization, a surrogate model is usually trained for each objective independently, and an acquisition function is adapted to overlook multiple models. Examples of such acquisition function are *expected hypervolume improvement* (EHI) [18] and *predictive entropy search* (PES) [24].

Chapter 4

The DGEMO Algorithm

In this chapter, we present our multi-objective Bayesian optimization approach that proposes batches of samples to evaluate in each iteration.

Let $\mathbf{f} : \mathcal{X} \to \mathbb{R}^m$ be a vector of m black-box objectives of form $f_j : \mathcal{X} \to \mathbb{R}$, where $\mathcal{X} \subset \mathbb{R}^d$. The input to our method is a small set of initial samples $X_0 \subset \mathcal{X}$, usually drawn by *Latin Hypercube Sampling* (LHS) [37], and the corresponding evaluations $Y_0 = \{\mathbf{f}(\mathbf{x}_i), \forall \mathbf{x}_i \in X_0\} \subset \mathbb{R}^m$. Our approach consists of three main components: building a surrogate model G_j for each objective f_j (Section 4.1), approximation of the Pareto set $\mathcal{P}_s$ and Pareto front $\mathcal{P}_f$ (Section 4.2), and finally a selection of next set of samples to evaluate (Section 4.3). We iterate over these three components to efficiently improve the Pareto-optimal solution. The approach is summarized in Algorithm 1.

4.1 Surrogate model

We use Gaussian process (GP) as a surrogate model to model each objective independently. First, the prior of GP model is characterized by the *mean function* $m : \mathcal{X} \to \mathbb{R}$ and the *kernel function* $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$. For a given input variable $\mathbf{x}$, the prior distribution of GP can be stated as $f(\mathbf{x}) \sim \mathcal{N}(m(\mathbf{x}), k(\mathbf{x}, \mathbf{x}))$. GP prior can be used to incorporate any prior beliefs about the objective functions if available, without depending on the input data, by choosing different mean function and kernel function.

Algorithm 1 DGEMO

Inputs: Design space $\mathcal{X}$; black-box objectives $\mathbf{f}(\mathbf{x}) = (f_1(\mathbf{x}), ..., f_m(\mathbf{x}))$; number of iterations n; number of initial samples k; batch size b.
Output: Pareto set $\mathcal{P}_s$ and Pareto front $\mathcal{P}_f$.
$X_0 \leftarrow \{\mathbf{x}_1, \ldots, \mathbf{x}_k\}, Y_0 \leftarrow \{\mathbf{f}(\mathbf{x}_1), \ldots, \mathbf{f}(\mathbf{x}_k)\}$ // initial samples drawn from LHS
for $i \leftarrow 0$ **to** n **do**
  Train surrogate models $G_j^{(i)}$ on X_i, Y_i for each objective f_j, $j \in \{1, \ldots, m\}$
  Define acquisition function $\tilde{f}_j^{(i)}$ from each $G_j^{(i)}$, $\tilde{\mathbf{f}}^{(i)}(\mathbf{x}) \leftarrow (\tilde{f}_1^{(i)}(\mathbf{x}), \ldots, \tilde{f}_m^{(i)}(\mathbf{x}))$
  Approximate Pareto set $\mathcal{P}_s^{(i)}$ and Pareto front $\mathcal{P}_f^{(i)}$ over $\tilde{\mathbf{f}}^{(i)}$
  Split points from $\mathcal{P}_s^{(i)}$ into diversity regions $\mathcal{D}_1^{(i)}, \ldots, \mathcal{D}_r^{(i)}$
  Select points $\mathbf{x}_1^{(i)}, \ldots, \mathbf{x}_b^{(i)}$ to evaluate from $\mathcal{D}_1^{(i)}, \ldots, \mathcal{D}_r^{(i)}$
  Evaluate and update $Y_{i+1} \leftarrow Y_i \cup \{\mathbf{f}^{(i)}(\mathbf{x}_1^{(i)}), \ldots, \mathbf{f}^{(i)}(\mathbf{x}_b^{(i)})\}$, $X_{i+1} \leftarrow X_i \cup \{\mathbf{x}_1^{(i)}, \ldots, \mathbf{x}_b^{(i)}\}$
Compute Pareto front $\mathcal{P}_f$ from points in Y_n and corresponding Pareto set $\mathcal{P}_s$

Without the loss of generality, we use mean function $m(\mathbf{x}) = 0$ and experiment with *Matern kernel functions* [43] that can capture a large variety of function properties. Secondly, we train a GP posterior to refine the prior model by maximizing the log marginal likelihood $\log p(\mathbf{y}|\mathbf{x}, \theta)$ on the available dataset $\{X, Y\}$, where θ denotes the parameters of the kernel function (e.g. $\theta = l$ for Matern kernel). Finally, the posterior distribution of GP is given in the form $f(\mathbf{x}) \sim \mathcal{N}(\mu(\mathbf{x}), \Sigma(\mathbf{x}))$, where the mean function is $\mu(\mathbf{x}) = m(\mathbf{x}) + \boldsymbol{k}\boldsymbol{K}^{-1}Y$, and covariance function $\Sigma(\mathbf{x}) = k(\mathbf{x}, \mathbf{x}) - \boldsymbol{k}\boldsymbol{K}^{-1}\boldsymbol{k}^T$, with $\boldsymbol{k} = k(\mathbf{x}, X)$ and $\boldsymbol{K} = k(X, X)$. To better explore the Pareto front based on this surrogate model, our Pareto front approximation algorithm (Section 4.2) makes use of the Jacobian and Hessian of the GP prediction $\mu(\mathbf{x}), \Sigma(\mathbf{x})$ w.r.t. $\mathbf{x}$. We provide the corresponding derivations in the section below.

4.1.1 Gaussian process derivatives

To better explore the Pareto front based on this surrogate model, our Pareto front approximation algorithm (see Section 4.2) makes use of the Jacobian and Hessian of

the GP prediction $\mu(\mathbf{x}), \Sigma(\mathbf{x})$ w.r.t. the input $\mathbf{x}$:

$$\frac{\partial \mu}{\partial \mathbf{x}} = \frac{\partial m}{\partial \mathbf{x}} + \frac{\partial \boldsymbol{k}}{\partial \mathbf{x}} \boldsymbol{K}^{-1} Y$$
$$\frac{\partial \mu}{\partial^2 \mathbf{x}} = \frac{\partial m}{\partial^2 \mathbf{x}} + \frac{\partial \boldsymbol{k}}{\partial^2 \mathbf{x}} \boldsymbol{K}^{-1} Y$$
$$\frac{\partial \Sigma}{\partial \mathbf{x}} = -\frac{\partial \boldsymbol{k}}{\partial \mathbf{x}} \boldsymbol{K}^{-1} \boldsymbol{k}^T - \boldsymbol{k} \boldsymbol{K}^{-1} \frac{\partial \boldsymbol{k}^T}{\partial \mathbf{x}}$$
$$\frac{\partial \Sigma}{\partial^2 \mathbf{x}} = -\frac{\partial \boldsymbol{k}}{\partial^2 \mathbf{x}} \boldsymbol{K}^{-1} \boldsymbol{k}^T - 2\frac{\partial \boldsymbol{k}}{\partial \mathbf{x}} \boldsymbol{K}^{-1} \frac{\partial \boldsymbol{k}^T}{\partial \mathbf{x}} - \boldsymbol{k} \boldsymbol{K}^{-1} \frac{\partial \boldsymbol{k}^T}{\partial^2 \mathbf{x}}$$

where $\boldsymbol{k} = k(\mathbf{x}, X)$, $\boldsymbol{K} = k(X, X)$, $\frac{\partial m}{\partial \mathbf{x}} = 0$ and $\frac{\partial m}{\partial^2 \mathbf{x}} = 0$ since we use $m(\mathbf{x}) = 0$.

In this thesis we use Matern 5/2 kernel [43] as default, defined as:

$$\boldsymbol{k} = \sigma_f^2 (1 + \sqrt{5}d + \frac{5}{3}d^2) \exp(-\sqrt{5}d) + \sigma_n^2$$

where $\sigma_f \in \mathbb{R}$ is a scaling parameter and $\sigma_n \in \mathbb{R}$ is a constant parameter, d denotes Euclidean distance $d(\mathbf{x}/l, X/l)$, and $l \in \mathbb{R}^d$ is the length-scale parameter of the kernel. The derivation of $\frac{\partial \boldsymbol{k}}{\partial \mathbf{x}}$ and $\frac{\partial \boldsymbol{k}}{\partial^2 \mathbf{x}}$ on this kernel is as follows:

$$\frac{\partial \boldsymbol{k}}{\partial \mathbf{x}} = -\frac{5}{3}\sigma_f^2 \exp(-\sqrt{5}d)(1 + \sqrt{5}d) d \frac{\partial d}{\partial \mathbf{x}}$$
$$\frac{\partial \boldsymbol{k}}{\partial^2 \mathbf{x}} = -\frac{5}{3}\sigma_f^2 \exp(-\sqrt{5}d)(-5d^2 (\frac{\partial d}{\partial \mathbf{x}})^2 + (1 + \sqrt{5}d)((\frac{\partial d}{\partial \mathbf{x}})^2 + d\frac{\partial d}{\partial^2 \mathbf{x}}))$$

We omit the derivation for derivatives of Euclidean distance d since it is well known. And extending the derivation to other kernels such as Matern 1/2, Matern 3/2 and RBF [43] kernels is straightforward.

4.2 Pareto front approximation

After obtaining a surrogate model G_j for each objective function f_j, we simply use the mean function of GP posterior as an acquisition function $\tilde{f}_j = \mu_j$. The next step of our algorithm then is to compute the Pareto front over all $\tilde{f}_j$. A core contribution to the efficacy of our algorithm comes from the Pareto front approximation approach, based on the method proposed in [47]. In typical MOO methods, discovering an

individual point that minimizes the set of functions $\{f_1(\mathbf{x}), \ldots, f_m(\mathbf{x})\}$, s.t. $\mathbf{x} \in \mathcal{X}$ can be challenging. However, once a single solution is discovered, locally searching around this point is easier. By solving a dual problem based on *KKT conditions* [25], the approach of [47] leverages this fact and derives a first-order approximation of the Pareto front. They efficiently discover large piecewise continuous regions of the Pareto front, rather than a sparse set of points obtained by random mutations from popular evolutionary algorithms. For more details, we refer the reader to [47].

In summary, the Pareto front approximation implements an iterative procedure consisting of three main steps. First, a *stochastic sampling* scheme is used to generate a set of random samples $\mathbf{x}_i \in \mathcal{X}$ perturbed from the best samples found so far, to avoid local minima and balance between exploration and exploitation of the design space neighborhoods. The second step implements a *local optimization* procedure that drives each sample $\mathbf{x}_i$ towards a local Pareto-optimal solution $\mathbf{x}_i^*$ with L-BFGS solver [33]. Different optimization directions are explored to cover diverse regions of the Pareto front. Finally, a *first-order approximation* of the Pareto front is extracted around $\mathbf{x}_i^*$, resulting in a dense set of solutions located on a $(m-1)$-dimensional manifold that approximates a continuous region on the Pareto front. The last step delivers the key advantage of this approach: once a single point in a Pareto set is found, it can be used to efficiently discover large Pareto-optimal regions in its neighborhood. In our experiments, we found that only 10 iterations of this procedure are enough to produce a high-quality approximation. One example that presents the resulting Pareto front approximation of a benchmark problem is shown in Figure 4-1. Note that due to the limited budget of evaluations, other algorithms waste most of the samples trying to expand a single region and struggle to escape the local minima, while ours manages to quickly discover all regions of the true Pareto front and focus most evaluations in these regions, covering almost the entire ground truth Pareto front (gray) in only 20 iterations of the algorithm with batch size 10.

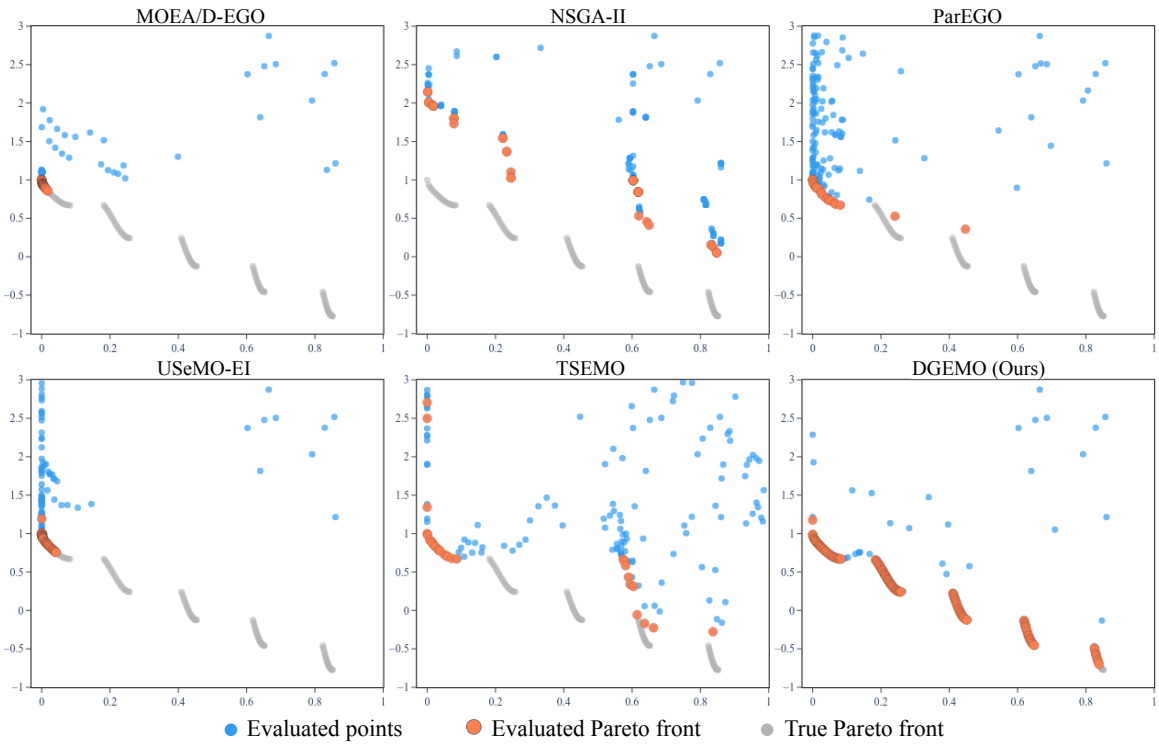


Figure 4-1: Comparison of evaluated points (blue and orange) performed by different algorithms on standard ZDT3 problem.

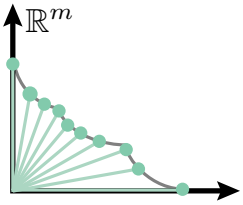
4.3 Batch selection strategy

The last part of our algorithm proposes a novel strategy for selecting a batch of samples to evaluate in the next iteration. We first present the approach for defining *diversity regions*; we then describe the selection strategy and corresponding algorithm.

4.3.1 Diversity regions

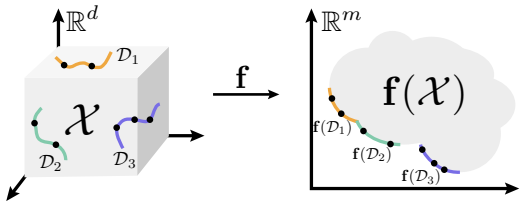
Starting from the Pareto front approximation, the goal is to group the optimal points based on their design properties and performance, resulting in several *diversity regions*. Standard MOO methods, such as NSGA-II, tend to output a Pareto front as a sparse set of points uniformly scattered around performance space, without considering the design space diversity. On the contrary, in our method, the Pareto front representation obtained from Section 4.2 has compact regions and better space coverage. From this representation, we can easily trace the neighboring points and

their correlation in both design and performance space. To solve the grouping problem, we rely on a data structure called *performance buffer*, implemented by [47]. The performance buffer sorts out the best performing points from the Pareto front approximation by storing them into an $(m-1)$-dimensional array discretized with hyperspherical coordinates. Assuming that all objectives are positive, any point that lies on the Pareto front will intersect a positive ray traced from the origin (see inset). The best performing points are selected as the points closest to the origin and sorted based on their hyperspherical coordinates. The buffer stores the performance and design space locations of each selected point, as well as the corresponding linear subspace of the design space extracted with the first-order approximation step. This way, the buffer keeps both the optimality information from the performance space and the closeness information from the design space. This data structure is compatible with a graph-cut method, which then extracts a sparse subset of optimal points grouped into k linear subspaces $\mathcal{D}_1, \dots \mathcal{D}_k$. The points are grouped based on the quality of their performance and their neighborhoods in the design space. We use these linear subspaces as our diversity regions.



4.3.2 Selection strategy

Our strategy involves two criteria: diversity measure and hypervolume improvement. Our diversity measure incorporates the knowledge from both design and performance space, and aims to evenly distribute selected samples among diversity regions (see inset). This measure then enforces the exploration of different regions of the Pareto front, while also considering diversity in the design space. It is especially important in the initial iterations when the uncertainty in the model is high and hypervolume improvement is often inaccurate. It further prevents the optimization from falling into local minima and overly exploiting one high-performing area, while neglecting other potentially promising regions.



Our selection strategy aims to maximize the hypervolume improvement while enforcing the samples to be taken from all diverse regions as uniformly as possible. These requirements can be written in the following form:

$$\arg\max_{X_B} \mathrm{HVI}(Y_B, \mathcal{P}_f) \quad \text{s.t.} \max_{1 \leq i \leq |\mathcal{D}|} \delta_i(X_B) - \min_{1 \leq i \leq |\mathcal{D}|} \delta_i(X_B) \leq 1 \tag{4.1}$$

where $X_B = \{\mathbf{x}_1, \ldots, \mathbf{x}_b\}$ is a set of b samples in a batch, $Y_B = \{\tilde{\mathbf{f}}(\mathbf{x}_1), \ldots, \tilde{\mathbf{f}}(\mathbf{x}_b)\}$, $\mathcal{P}_f$ is the current Pareto front, functions $\delta_i(.)$ are defined for each region $\mathcal{D}_i \in \mathcal{D}$ as a number of elements $\mathbf{x}_j$ from X_B that belong to the region $\mathcal{D}_i$.

4.3.3 Selection algorithm

The optimization problem defined in Equation 4.1 can be solved combinatorially. Nevertheless, it would be extremely computationally expensive. Instead, we implement a greedy approach. We select a point $\mathbf{x}_1$ with the largest hypervolume improvement, add $\mathbf{x}_1$ to the batch, and add $\tilde{\mathbf{f}}(\mathbf{x}_1)$ to the current Pareto front $\mathcal{P}_f$. We then search for the next point with the largest hypervolume improvement, which does not belong to the same region as $\mathbf{x}_1$. We repeat this process while avoiding all the regions from which the points were already selected until all regions are covered. If more points are still needed for the batch, we reset the counter of covered regions and repeat the same process. This way we obtain evenly distributed points around regions, while also aiming to maximize the hypervolume improvement. The complete algorithm is presented in Algorithm 2.

Algorithm 2 Batch Selection Algorithm

Inputs: Current Pareto front $\mathcal{P}_f$; batch size b; candidate points split into regions $\mathcal{D}_1, \ldots, \mathcal{D}_r$; surrogate objectives $\tilde{\mathbf{f}}(\mathbf{x}) = (\tilde{f}_1(\mathbf{x}), ..., \tilde{f}_m(\mathbf{x}))$.
Output: Batch of selected samples X_B.
$S \leftarrow \{1, 2, ..., r\}$ // set of available regions to choose
for $i \leftarrow 1$ **to** b **do**
  $h_{max} \leftarrow -\infty, s_{max} \leftarrow 0, \mathbf{x}_{max} \leftarrow null$ // record the sample with maximal HVI
  for each $s \in S$ **do**
    for each $\mathbf{x} \in \mathcal{D}_s$ **do**
      $h \leftarrow \text{HVI}(\tilde{\mathbf{f}}(\mathbf{x}), \mathcal{P}_f)$
      if $h > h_{max}$ **then** $h_{max} \leftarrow h, s_{max} \leftarrow s, \mathbf{x}_{max} \leftarrow \mathbf{x}$
  $X_B \leftarrow X_B \cup \{\mathbf{x}_{max}\}$, $\mathcal{P}_f \leftarrow \mathcal{P}_f \cup \{\tilde{\mathbf{f}}(\mathbf{x}_{max})\}$ // aggregate solutions
  $\mathcal{D}_{s_{max}} \leftarrow \mathcal{D}_{s_{max}} - \{\mathbf{x}_{max}\}$, $S \leftarrow S - \{s_{max}\}$
  if $S = \emptyset$ **then** $S \leftarrow \{1, 2, ..., r\}$ // reset the counter of visited regions

Chapter 5

Experiment Results

We now compare our algorithm to the relevant state-of-the-art methods on both synthetic test functions and real-world benchmark problems.

5.1 Benchmark problems

First, we conduct experiments on 13 synthetic multi-objective test functions including ZDT1-3 [60], DTLZ1-6 [16], OKA1-2 [40], VLMOP2-3 [53], which are widely used in previous literature. The experiments include problems with 2 and 3 objectives, and the number of design variables varying from 2 to 7. Second, we adopt 7 real-world engineering design problems presented in RE problem suite [52], which are: four bar truss design, reinforced concrete beam design, hatch cover design, welded beam design, disc brake design, gear train design, and rocket injector design. We use names RE1-7 to refer to these problems in plots and further text.

In this section, we briefly introduce the properties of each problem, including the dimensions of the design space $\mathcal{X} \subset \mathbb{R}^d$ and performance space $\mathbf{f}(\mathcal{X}) \subset \mathbb{R}^m$, and the reference points we use for calculating the hypervolume indicator. The problem descriptions for 13 synthetic functions and 7 real-world problems are shown in Table 5.1 and Table 5.2 respectively. For all functions that can work with arbitrary dimensions, we use 6 variables and 2 objectives for consistency in testing. We perform 10 independent test runs with 10 different random seeds for each problem on each

algorithm. For each test run of one problem, we use the same initial set of samples for every algorithm, which is generated by Latin hypercube sampling [37] using a same random seed. To have a fair comparison, we simply set the reference point $\mathbf{r} \in \mathbb{R}^m$ as a vector containing the maximum value of each objective over the initial set of samples $\{\mathbf{x}_1, ..., \mathbf{x}_k\}$:

$$\mathbf{r} = (\max_{1 \leq i \leq k} f_1(\mathbf{x}_i), ..., \max_{1 \leq i \leq k} f_m(\mathbf{x}_i)).$$

Table 5.1: Description of synthetic functions.

Name	d	m	$\mathbf{r}$
ZDT1	6	2	(0.9902, 6.3936)
ZDT2	6	2	(0.9902, 7.7158)
ZDT3	6	2	(0.9902, 6.5464)
DTLZ1	6	2	(360.7570, 343.4563)
DTLZ2	6	2	(1.7435, 1,6819)
DTLZ3	6	2	(706.5260, 746.2411)
DTLZ4	6	2	(1.8111, 0.7776)
DTLZ5	6	2	(1.7435, 1.6819)
DTLZ6	6	2	(5.7482, 5.6523)
OKA1	2	2	(7.4051, 4.3608)
OKA2	3	2	(3.1315, 4.6327)
VLMOP2	6	2	(1.0, 1.0)
VLMOP3	2	3	(8.1956, 53.2348, 0.1963)

Table 5.2: Description of real-world problems.

Name	Description	d	m	$\mathbf{r}$
RE1	Four bar truss design [9]	4	2	(2967.0243, 0.0383)
RE2	Reinforced concrete beam design [1]	3	2	(703.6860, 899.2291)
RE3	Hatch cover design [1]	2	2	(5885.4870, 5.5063)
RE4	Welded beam design [44]	4	3	(202.8569, 42.0653, 2111643.6209)
RE5	Disc brake design [44]	4	3	(6.1356, 6.3421, 12.9737)
RE6	Gear train design [15]	4	3	(6.6764, 59.0, 0.4633)
RE7	Rocket injector design [20]	4	3	(0.8136, 0.8889, 0.9799)

5.2 Algorithms and implementation

We compare our algorithm to several baseline algorithms described in Chapter 2: NSGA-II [14], ParEGO [28], MOEA/D-EGO [58], TSEMO [8], and USeMO-EI [4]. We extend ParEGO and USeMO-EI from their original sequential selection to batch selection for batch evaluation scenario; see Appendix A for details of the extension. We implement and compare all algorithms in our Python codebase, built upon pymoo [6], a state-of-the-art Python framework for multi-objective optimization. Our code is be released open-source with reproducibility guarantee.

5.3 Comparison metric

To compare the performance of different algorithms, we mainly use *hypervolume indicator* – the most popular comparison metric for multi-objective optimization problems [45]. This indicator measures the quality of the set of discovered solutions. We monitor the increase of the hypervolume indicator (Eq. 3.1) over the number of evaluations performed. For comparison fairness, we use the same reference point and initial set of samples for every algorithm. See Section 5.1 for more details on initial samples and reference points used for each problem.

Additionally, we also compare the performance of DGEMO and other baseline algorithms using *generational distance (GD)* [53] and *inverted generational distance (IGD)* [10] as metrics, which are also very popular metrics in MOO literature besides hypervolume. These metrics measure the Euclidean distance between the Pareto front approximation found by the algorithm and the true Pareto front in different ways, thus the lower the better. But since most our benchmark problems do not have an analytical solution of the true Pareto front, in practice we approximate the true Pareto front by merging all the Pareto front approximations found by all algorithms using all random seeds.

5.4 Results and discussion

The selected test problems serve to validate the quality of the method when dealing with functions that are concave, convex, disconnected, multimodal, and of diverse design and performance space complexity. Our approach shows consistent top performance, compared to other algorithms that oscillate on different types of problems. DGEMO also proves to be particularly good on the synthetic functions mimicking the real-world data (OKA, VLMOP) and the actual real-world problems (RE).

5.4.1 Hypervolume indicator

Figure 5-1 presents the comparison of the hypervolume indicator w.r.t. evaluation number. For every algorithm, we run every experiment with 10 different random seeds and the same 50 initial samples. In these experiments, we use a batch of 10 samples in each iteration and ran 20 iterations in total. All hyperparameters used are presented in Appendix A. The analysis on computational complexity can be found in Appendix B. We also conduct experiments using different batch sizes, including 1, 2, 4, 5 and 20, and our algorithm still consistently outperforms others. See Appendix C for more results. The Pareto front approximation approach presented in Section 4.2 enables our method to work well even with the small batch size (including a sequential case with batch size 1). However, while other methods are unstable and can vary the performance quality when the batch size increases, our method always exhibits superior and stable results thanks to the diversity-guided selection strategy.

5.4.2 GD and IGD indicators

Figure 5-2 and 5-3 presents the comparison of the generational distance (GD) and inverted generational distance (IGD) indicator w.r.t. evaluation number, respectively. In these experiments, we use a batch of 10 samples in each iteration and ran 20 iterations in total. The results also show that DGEMO performs the best across almost all the benchmark problems under either comparison metric.

5.4.3 Model prediction error

In experimental design problems, aside from searching for an optimal set of solutions, it is often desirable to obtain a reliable prediction of the performance of untested samples. Hence, the quality of the surrogate model and the Pareto front approximation can be an important criterion in choosing the algorithm to work with. In Table 5.3 we show the average prediction error of all evaluated points produced with our approach and other MOBO baseline algorithms (NSGA-II does not have a surrogate model so comparison is infeasible). The prediction error is calculated by $\|\mathbf{f}(\mathbf{x}) - \tilde{\mathbf{f}}(\mathbf{x})\|$ as the Euclidean distance between the surrogate prediction and the actual performance. The scale of the error varies across different problems due to the problem definition, as some objectives have extremely high but valid values. It turns out that besides the superior performance of the hypervolume metric, DGEMO also has the lowest prediction error among the majority of the problems.

Table 5.3: Averaged surrogate prediction error of points proposed by different algorithms, including DGEMO, on all benchmark problems.

Method	ZDT1	ZDT2	ZDT3	DTLZ1	DTLZ2	DTLZ3	DTLZ4	DTLZ5	DTLZ6	OKA1
ParEGO	7.87	2.29	34.9	1.76e+04	6.95	3.83e+04	**30.1**	6.93	304	679
MOEA/D-EGO	76	8.11	106	2.09e+04	5.75	4.09e+04	124	5.75	6.37e+03	3.63e+03
TSEMO	65.9	5.48	331	1.78e+05	41.6	4.12e+05	607	41.6	1.43e+03	544
USeMO-EI	352	1.39	420	2.25e+04	22.3	5.54e+04	92.2	22.3	3.08e+04	1.55e+03
DGEMO (Ours)	**5.25**	**0.933**	**11.6**	**8.24e+03**	**1.59**	**1.5e+04**	71.6	**1.59**	**275**	**233**

Method	OKA2	VLMOP2	VLMOP3	RE1	RE2	RE3	RE4	RE5	RE6	RE7
ParEGO	**58.8**	**5.4**	61.6	5.93	2.86e+16	98.6	**1.95e+08**	575	292	0.597
MOEA/D-EGO	297	9.22	69.4	10.2	1.23e+15	264	1.19e+09	284	1.64e+03	1.26
TSEMO	641	31.9	284	153	1.19e+17	1.32e+03	1.47e+10	229	1.33e+03	0.734
USeMO-EI	1.21e+03	17.5	115	16.4	7.74e+15	386	6.72e+09	**114**	976	1.14
DGEMO (Ours)	140	6.26	**21**	**3.09**	**3.25e+11**	**92.7**	2.13e+08	1.22e+03	**288**	**0.343**

5.4.4 Discussion and analysis

Given a trained surrogate model, unlike the others, our approach makes use of the Jacobian and Hessian information of this model in Pareto front approximation. We observe that DGEMO needs fewer iterations to stabilize the prediction, and greatly improves the hypervolume quality already in the first few iterations. Experiments that have a budget of less than 20 iterations would highly benefit from this property of our approach. In addition, DGEMO explores and provides additional information on

diverse regions of Pareto-optimal solutions based on their properties and performance. For many complex real-world design problems, there are obvious clusters in the Pareto-optimal solutions. It is therefore desirable to fully explore these diverse regions of interest for a more comprehensive understanding of the problem and having a wider range of solutions.

We take the rocket injector design problem (RE7) [1] as an example. In this problem, the primary objectives are performance and material sustainability. Specifically, the goal is to minimize the combustion length (X_{cc}) for better size and efficiency of the combustor, while at the same time minimizing the maximum temperature of injector face (TF_{max}) and post tip (TT_{max}) for a longer material life. The design variables consist of hydrogen flow angle (α), hydrogen area ($\triangle$HA), oxygen area ($\triangle$OA), and oxidizer post tip thickness (OPTT).

The Pareto-optimal solutions of this problem are grouped in several regions in design space, and these groups correspond to different patches on the Pareto front, see [20]. This phenomenon shows that for the rocket injector structure, there is no single optimal solution that guarantees the optimal performance on all the objectives involved. Instead, there are different design patterns that lead to optimal properties with different trade-offs. To understand the deeper connection between design space and performance space, how certain design variables lead to Pareto optimality, and the pattern difference between these groups, we explore diversity regions detected by our algorithm. Figure 5-4 presents a visualization of the 3-dimensional perofrmance space, where each diversity region is represented with a different color.

From Figure 5-4 we can see different clusters are indeed responsible for different Pareto-optimal regions on the Pareto front. By analyzing the design variables of the Pareto set, we list several interesting observations illustrated as follows:

- The widest blue region of the Pareto front includes solutions with the highest TT_{max}. This region is the only region with Pareto-optimal points that have the OPTT > 0, and these points are located in the upper part of the blue region.

[1] The original rocket injector design problem [20] has four objectives defined. However, according to their analysis, two of them have strong correlation, hence we simplify it as a three objective problem as also suggested in [20].

We note that larger OPTT design variable leads to larger TT_{max}. Hence, the oxidizer thickness has the most important contribution to the longer material life of the post tip and solely contributes to this objective. In addition, an important design characteristic of this region is that the oxygen area is always 0, while the other two design variables (hydrogen flow angle and hydrogen area) vary between 0 and 1. Varying the values of α and $\triangle$HA, controls the trade-off between X_{cc} and TF_{max}.

- When hydrogen flow angle and oxygen area are maximally utilized ($\alpha = 1$ and $\triangle\mathrm{OA} = 1$), TT_{max} has the minimal optimal value (gray region). By ranging the value of $\triangle$HA from 0.1 to 1, X_{cc} increases while TF_{max} decreases, respectively. Hence hydrogen area controls the trade-off between the TF_{max} and X_{cc}, similar to the pattern we found in blue region.

- When the hydrogen area is maximal ($\triangle\mathrm{HA} = 1$), oxygen area is very high ($\triangle\mathrm{OA} > 0.9$), oxydizer post tip is not in use ($\mathrm{OPTT} = 0$), and the hydrogen flow angle ranges from 0.5 to 1, the changes in TF_{max} are negligible, while TT_{max} decreases from 0.1 to -0.3 as α increases (purple region). For applications targeting a design with minimal face temperature, we found the design variables should have maximal hydrogen area and oxygen area, at the same time with minimal hydrogen flow angle and zero OPTT, as shown in the green region.

- In general, among the Pareto-optimal solutions, a small flow angle indicates low face temperature and high tip temperature. Increasing the flow angle leads to an increase in the face temperature and a decrease in the combustion length. Large hydrogen area corresponds to low tip and face temperature but long combustion length. Most of the Pareto-optimal solutions have a relatively low tip thickness.

As a result, by analyzing the regions of on the Pareto set we can gain more understanding about the nature of the problem. In many complex real-world problems, such as this rocket injector design, the Pareto-optimal solutions are grouped in different regions and present different performance trade-offs. The DGEMO algorithm can

discover these diverse sets of solutions by directly taking the diversity in both design space and performance space into account. From the solutions found by our algorithm, we can easily extract the representative design patterns from these regions of Pareto-optimal designs, and potentially combine the knowledge concluded from these patterns with prior knowledge to discover an even better Pareto set.

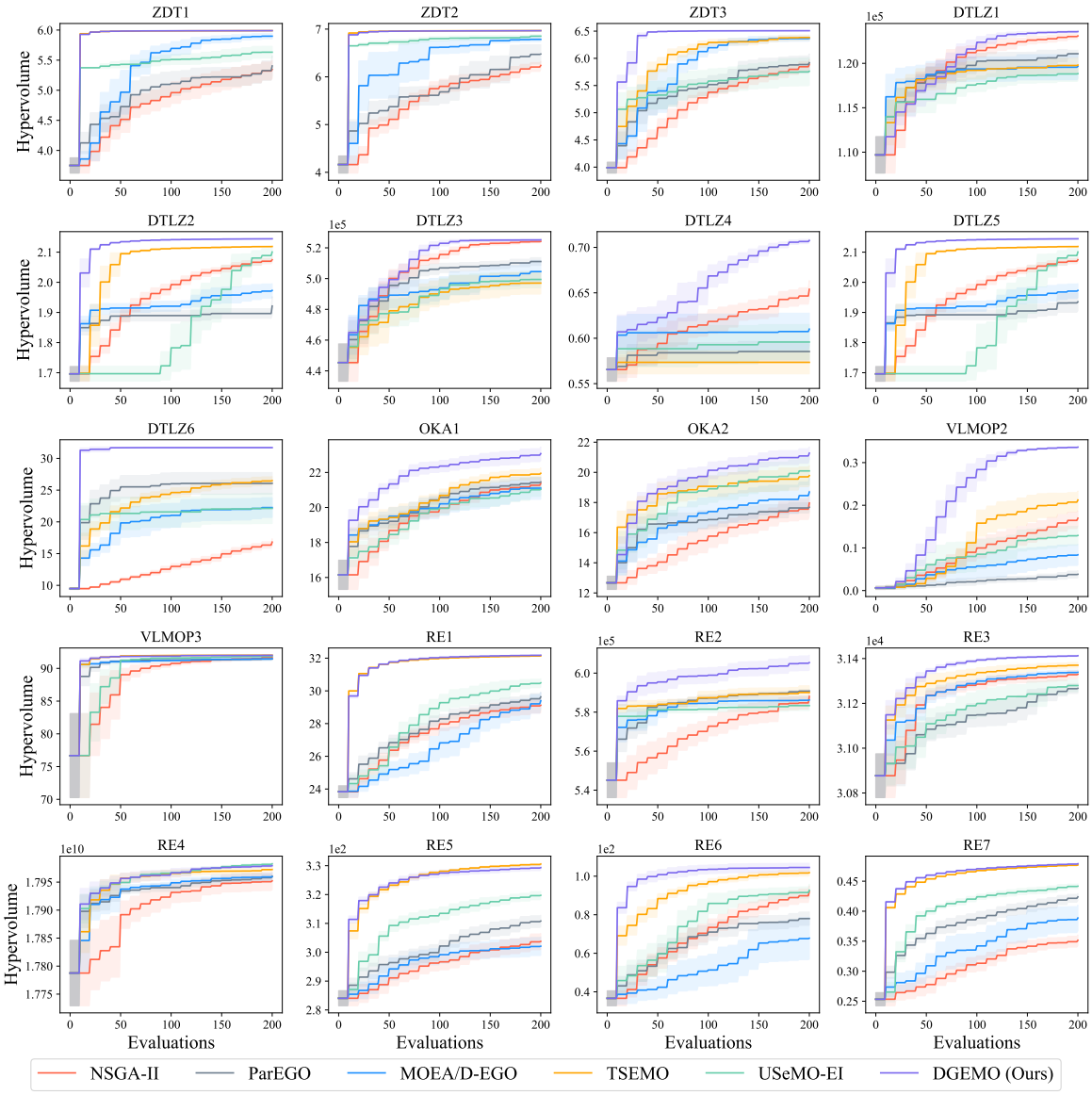


Figure 5-1: Hypervolume indicator of different algorithms on all problems with batch size as 10, shown with respect to the different number of function evaluations. The curve is averaged over 10 different random seeds and the variance is shown as a shaded region.

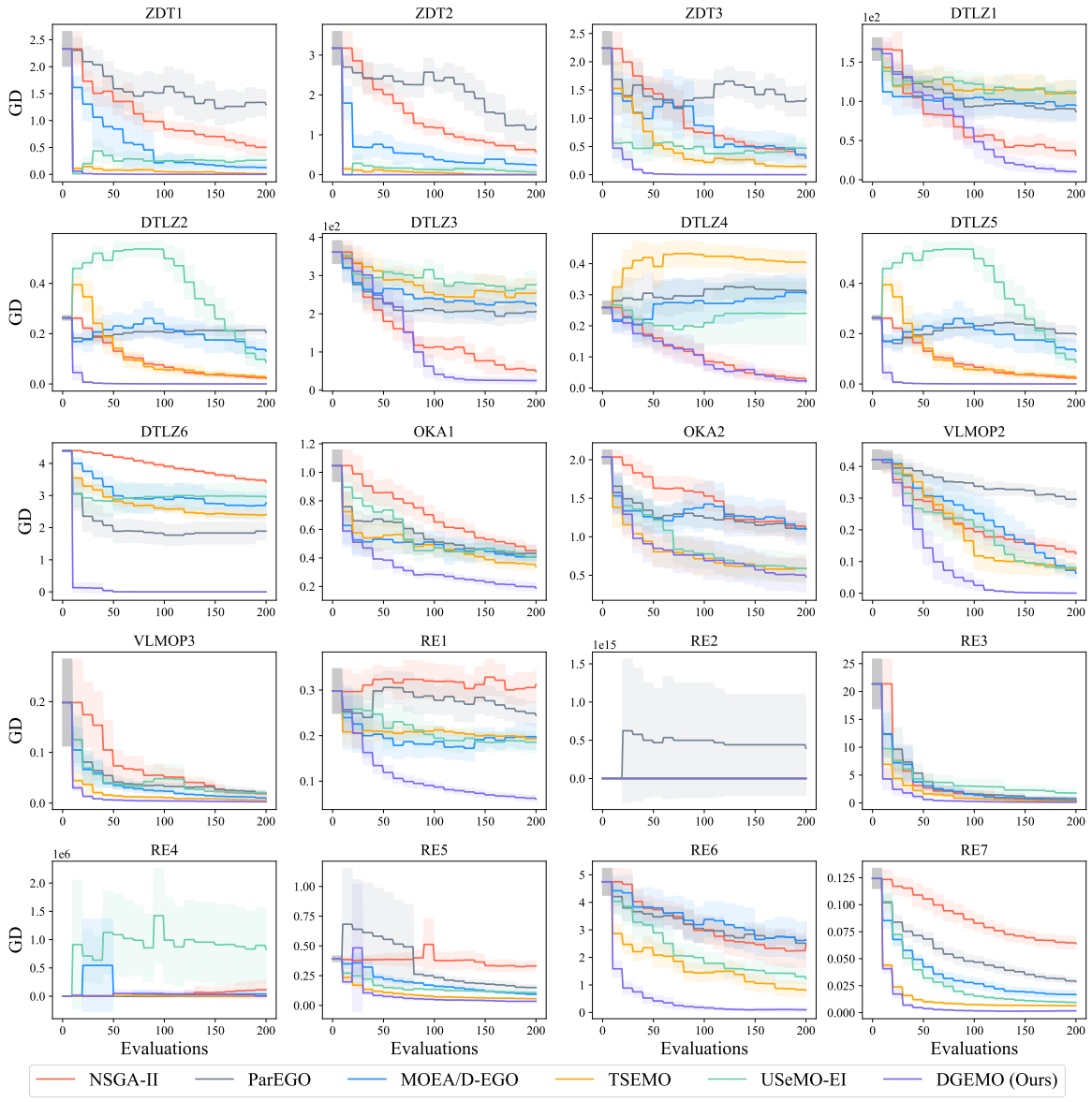


Figure 5-2: Generational distance of different algorithms on all problems with batch size as 10, shown with respect to the different number of function evaluations.

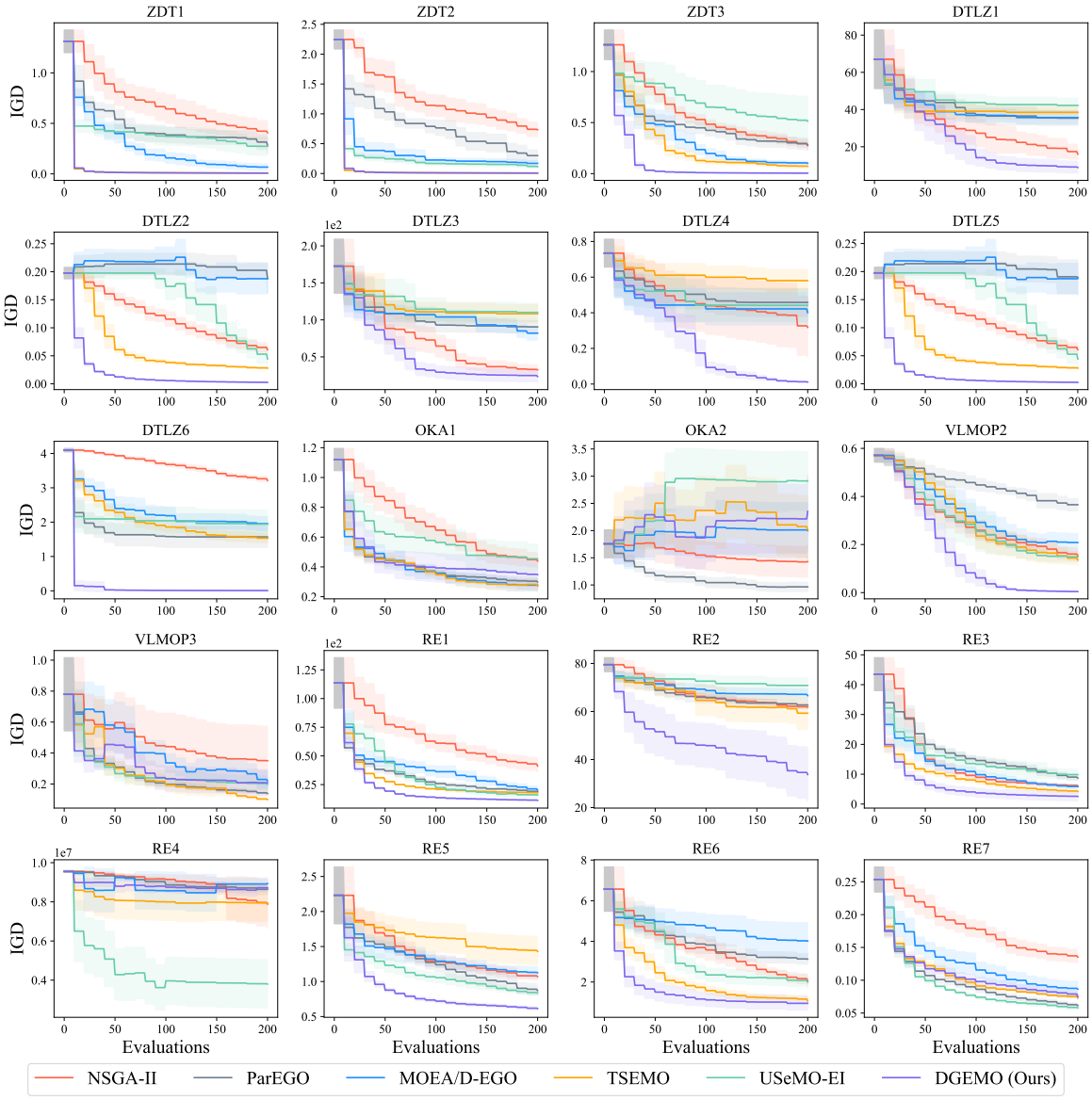


Figure 5-3: Inverted generational distance of different algorithms on all problems with batch size as 10, shown with respect to the different number of function evaluations.

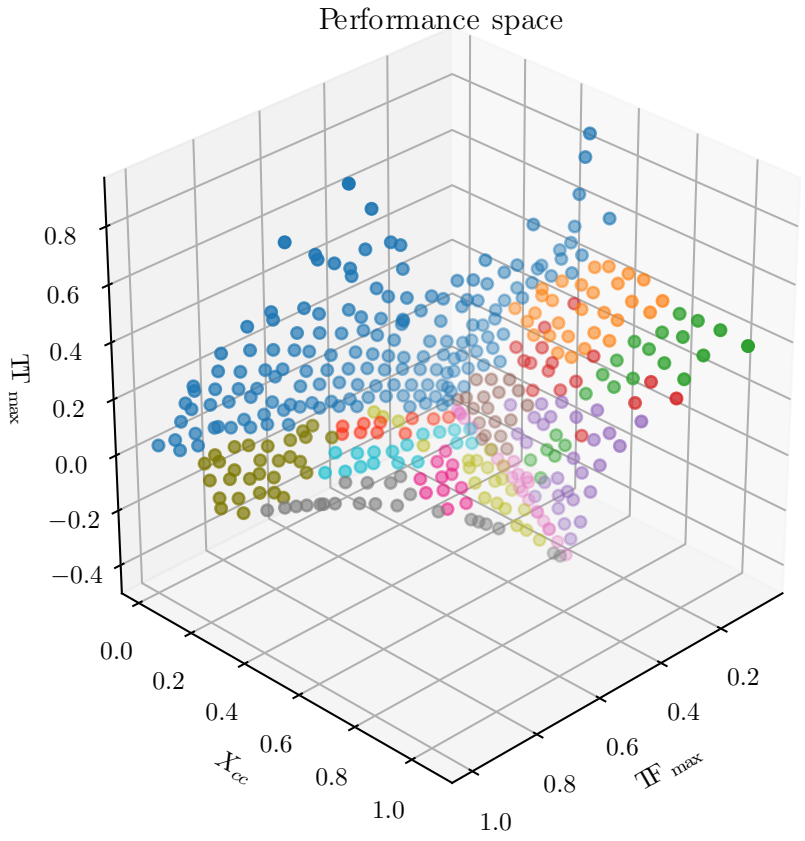


Figure 5-4: Performance space visualization of Pareto-optimal solutions and their diversity regions found by DGEMO on rocket injector design (RE7) problem (in 3D visualization, the opacity of points represent the depth from current point of view).

Chapter 6

The AutoOED Platform

AutoOED is an open-source platform for multi-objective optimization problems with a restricted budget of test samples. The platform automatically guides the design of experiments to be evaluated and quickly leads to the best performing designs. AutoOED is based on the concept of multi-objective Bayesian optimization (MOBO) and is entirely implemented in Python. The key features of this platform include:

- **Intuitive GUI**: An easy-to-use graphical user interface (GUI) is provided to directly visualize and guide the optimization progress and facilitate the operation for users with little or no experience with coding, optimization, or machine learning. The GUI is built using the Tkinter package with a model-view-controller (MVC) architecture.

- **Modular structure**: A highly modularized codebase and built-in visualization enable easy extensions and replacements of MOBO algorithm components. The platform can serve as a testbed for machine learning researchers to easily develop and evaluate their own MOBO algorithms.

- **Data-efficient experimentation**: As the platform aims to solve problems with expensive to evaluate or black-box objective functions, the number of experiments is limited to an order of several dozens. The platform employs an optimization strategy that rapidly advances the Pareto front with a small set of evaluated experiments.

- **Sequential and batch evaluations**: A standard feature supports evaluating a single experiment in each optimization iteration. To further reduce the optimization time, the platform parallelizes the evaluation process by enabling synchronous and asynchronous batch evaluations. Asynchronous batch evaluations are instrumental when multiple workers are running experiments, but their evaluations drastically vary in time.

- **Automation of experiment design**: The platform is designed for straightforward integration into an automatic experiment design optimization pipeline. The pipeline may include both physical experiments (see example at `https://tinyurl.com/autooed-phys`) and an expensive simulation setup (see example at `https://tinyurl.com/autooed-sim`).

- **Distributed usage**: Besides the standard single-user version, to facilitate the platform's use for a single design process by several users, we provide a distributed team version based on a centralized database. The database shares the optimization status to different roles in the experiment team (managers, scientists, technicians) and provides them with different tools for controlling and contributing to the experimental design process.

6.1 Overview

To automate the optimization process and guide the experiments, AutoOED is based on the principles and algorithms of multi-objective Bayesian optimization (MOBO). MOBO is a data-driven approach that attempts to learn the black-box objective functions from available data and find Pareto-optimal solutions in a data-efficient manner. AutoOED outputs the whole palette of designs on the Pareto front, enabling the user to choose the trade-off most suited for their application. In addition to the set of optimal solutions, our platform's final product is the learned prediction models of the unknown objectives. The prediction provides the users with more insights into the potential outcomes of experiments. It helps them better understand the optimization

problem to make informed decisions and guide the optimization process towards their preference.

As a full-stack software, the overall workflow of AutoOED from the front-end to the back-end is presented in Figure 6-1. Given an optimization problem, if the evaluation program is available, AutoOED can automatically guide the experimentation by alternating between optimization and evaluation. In this case, an optimization process starts first. The scheduler then orders evaluations in an asynchronous way, and the optimization restarts once the evaluations from the currently requested batch are all done. This process repeats until some stopping criterion is met and the evaluated results are automatically recorded in the database. Otherwise, if the evaluation program does not exist, users can still manually do the evaluations and enter results into the database.

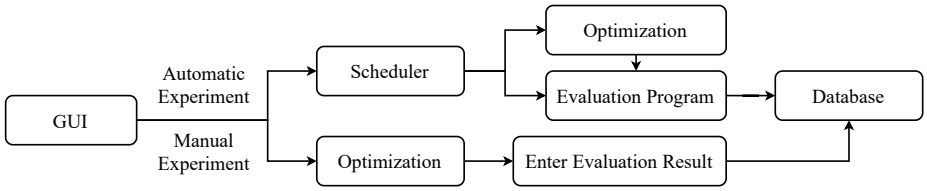


Figure 6-1: The overall workflow of the platform design.

6.2 Modular code structure

Multi-objective Bayesian optimization typically consists of four core components: (*i*) an inexpensive surrogate model for the black-box objective functions; (*ii*) an acquisition function that defines sampling from the surrogate model and trade-off between exploration and exploitation of the design space; (*iii*) a multi-objective optimization solver to approximate the Pareto set and front; (*iv*) a selection strategy that proposes a single or a batch of experiments to evaluate next. These four components (see Figure 6-2) are implemented as core and independent building blocks of the AutoOEDs, making it highly modularized and easy to develop new algorithms and modules. A list of supported surrogate models, acquisition functions, solvers, and selection strategies can be found at the API reference section of our documentation.

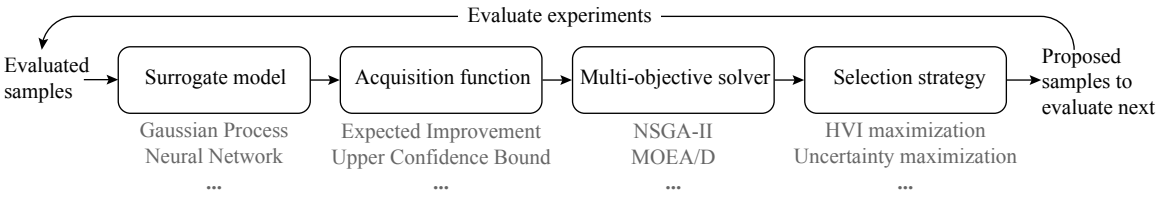


Figure 6-2: Algorithmic pipeline and core modules. The platform follows the multi-objective Bayesian optimization pipeline. A user can choose an approach for each module.

6.3 Supported algorithms and features

We provide several popular and state-of-the-art multi-objective Bayesian optimization methods, including DGEMO [34], TSEMO [8], USeMO [4], ParEGO [28]. To the best of our knowledge, DGEMO exhibits state-of-the-art performance for data-efficient, multi-objective problems with batch evaluations. Implemented algorithms also include NSGA-II [14] and MOEA/D [57], as they are some of the most widely used multi-objective evolutionary algorithms. Users can select an algorithm from this library that best fits the characteristics of their physical system or the optimization goals.

Additional features of the platform support: continuous, discrete, binary, categorical, and mixed design variables; linear and non-linear constraints handling in the design space; sequential and batch evaluations; synchronous and asynchronous batch evaluations. In the synchronous case, the algorithm starts a new iteration when all the requested evaluations from one batch are completed. The asynchronous batch evaluations are instrumental when evaluations drastically vary in time. To be more time-efficient, the asynchronous approach updates a model and requests a new evaluation as soon as one batch sample is completed. Our platform is especially beneficial for applications in areas such as finance, organization optimization, materials science, life sciences, chemistry, robotics, and the pharmaceutical industry, where the experiments are long (e.g., days or weeks) but can easily be carried out in parallel.

6.4 Graphical User Interface (GUI) and distributed usage

The GUI guides the user through a set of simple steps to configure the problem, such as the number of design and performance parameters, the parameter ranges and constraints, parallelization settings, and selection of the optimization algorithm. Other functional features include real-time display of design and performance space, exporting of the database and statistics, and support for linking custom evaluation scripts. We provide a manual GUI for entering the experiment data and example scripts that showcase how to interface the worker with simulation scripts (e.g., MATLAB structural analysis toolbox, see `https://tinyurl.com/autooed-sim`) or automated robotic platforms that perform experiments (see examples at `https://tinyurl.com/autooed-phys`).

For additional flexibility, a single experiment can be managed by several users from different computers. A distributed team version is based on a centralized database, providing secure data storage and retrieval. Users can be assigned different roles (scientist, technician, manager), and they will have different access and tools for controlling the experimental design process. This version helps the team collaborate more efficiently across the globe.

Chapter 7

Conclusion and Future Work

We introduced a novel multi-objective Bayesian optimization method named DGEMO that allows evaluation of batches of samples in each iteration. The selection of batch samples takes into consideration the diversity of samples in both design and performance space, making it suitable for physical experiments and many real-world problems. The key advantage of our approach lies in the Pareto front approximation algorithm and the batch selection strategy that are, to the best of our knowledge, different from any previously published work. We performed extensive tests on both synthetic test functions and real-world problems. We found that our approach significantly outperforms other methods in most of the test problems, and in others performs equally well as the best-scoring state-of-the-art algorithm.

While Gaussian processes are good at handling a certain amount of noise in the data, examining the effects of noise on the system in more detail is an interesting avenue for future research. Also, integrating DGEMO to an automated experimental lab setup to control the development and discovery of best performing experiments is an exciting new research direction. To this end, we proposed AutoOED, which is released under the MIT license as a free and open-source platform that can be obtained at `https://github.com/yunshengtian/AutoOED`. This thesis presents only a brief overview of the platform and its features. Comprehensive documentation and examples are available at `https://autooed.readthedocs.io`. The official webpage, as well as the past and ongoing projects using AutoOED, can be found at `https://autooed.org`.

Appendix A

Hyperparameters

For fair comparison among different algorithms, we try to use the same set of common hyperparameters as much as possible. The common hyperparameters are presented in Appendix A.1 and the algorithm-specific hyperparameters are presented in Appendix A.2. Here we list the important hyperparameters used in the implementation, and more details could be found in the code.

A.1 General hyperparameters

Surrogate model We use the same Gaussian process model as a surrogate for all experiments. We use zero mean function and anisotropic Matern 5/2 kernel. The corresponding hyperparameters are specified in Table A.1, which are suggested by TSEMO.

Table A.1: GP hyperparameters.

parameter name	value
initial l	$(1, ..., 1) \in \mathbb{R}^d$
l range	$(\sqrt{10^{-3}}, \sqrt{10^3})$
initial σ_f	1
σ_f range	$(\sqrt{10^{-3}}, \sqrt{10^3})$
initial σ_n	10^{-2}
σ_n range	$(e^{-6}, 1)$

Multi-objective evolutionary algorithm MOEA/D-EGO, TSEMO, USeMO-EI share the same NSGA-II solver using simulated binary crossover [12] and polynomial mutation [13] for finding the Pareto front of acquisition functions. The initial population is obtained from the best current samples determined by non-dominated sort [14]. The other hyperparameters are specified in Table A.2.

Table A.2: NSGA-II hyperparameters.

parameter name	value
population size	100
number of generations	200
crossover η_c	15
mutation η_m	20

We also use the baseline algorithm NSGA-II to compare with other MOBO algorithms. For that, we use the same hyperparameters for crossover η_c and mutation η_m as stated in Table A.2, but the population size is set to the batch size and the number of generations is equivalent to the number of algorithm iterations.

A.2 Algorithm-specific hyperparameters

ParEGO Since ParEGO is a single-point MOBO method, we extend it to the batch setting by using b random scalarization weights in each iteration, where b is the batch size. We use Chebyshev scalarization [38] and CMA-ES algorithm [23] for solving the scalarized single-objective problem with $\sigma = 0.5$ as initial standard deviation.

MOEA/D-EGO We use a similar implementation as the original MOEA/D-EGO, except that we remove the FuzzyCM. Nowadays it is already computationally efficient to train the Gaussian process model and directly use it for prediction, rather than relying on relatively faster but less accurate approximation methods. Hence, the performance of our implementation could potentially be relatively higher than the original implementation thanks to the accuracy gain of removing the FuzzyCM. We

use simulated binary crossover and polynomial mutation for MOEA/D, and the other hyperparameters used are described in Table A.3.

Table A.3: MOEA/D hyperparameters.

parameter name	value
number of reference directions	100
number of generations	200
number of neighbors	20
neighbor mating probability	0.9
crossover η_c	20
mutation η_m	20

TSEMO For TSEMO, we use mostly the same set of hyperparameters used in its original implementation. For spectral sampling we use 100 points.

USeMO-EI Since USeMO-EI is a single-point MOBO method, we extend it to the batch setting by selecting top-b points with maximal uncertainty, where b is the batch size.

DGEMO Most of the distinct hyperparameters of DGEMO come from the multi-objective optimization algorithm [47], as presented in Table A.4. In this MOO algorithm, the initial population is also obtained from the best current samples determined by non-dominated sort. Generally DGEMO is robust to the hyperparameter choices and in practice mildly changing the values of these hyperparameters wouldn't affect the final performance of DGEMO too much. For potentially slightly better performance, the number of buffer cells and the number of grid samples on local manifold could be increased for obtaining more candidate solutions to select, at the cost of spending more computation time. The label cost for graph cut is a key hyperparameter that controls the number of diversity regions, which could be adjusted case by case. Higher cost leads to less diversity regions. Even though we use the same label cost across all the benchmark problems, which is obviously

sub-optimal, DGEMO still outperforms other baseline algorithms. More suggestions on hyperparameter choices could be found in the original paper of this algorithm[47].

Table A.4: DGEMO hyperparameters.

parameter name	value
number of buffer cells	100 for 2-dim, 1000 for higher-dim
max number of samples in each cell	10
buffer origin	$(0, ..., 0) \in \mathbb{R}^m$
δ_b	0.2
δ_p	10
δ_s	0.3
label cost for graph cut	10
number of grid samples on local manifold	100

Appendix B

Computational Complexity

B.1 DEGMO algorithm complexity

Since our implementation is based on several advanced open-source Python packages doing the mathematical calculation (such as NumPy [41] and SciPy [54]) and highly memory-efficient, there is very a small memory occupation when running DGEMO algorithm. Here we mainly analyze the time complexity of our algorithm from three stages.

Surrogate model fitting It is well known that Gaussian process fitting is of $O(N^3)$ time complexity, where N is the size of the current dataset. However, in our case, N is usually not a big number (up to several hundred) since we are focusing on a problem setting with relatively few data available, and that is the common scenario in applications of Bayesian optimization.

Multi-objective optimization In the algorithm of [47], at each generation, for each individual in the population, the computationally expensive operations are the following four components: applying L-BFGS [33] for local optimization, SLSQP [29] for solving KKT dual variables, computing exploration directions by taking the null space of a matrix (see Equation 3 of [47]), and GP prediction on neighboring grid samples. Specifically, L-BFGS operates at $O(Md)$ time complexity, where M is a

relatively small number of algorithm updates and d is the dimension of design variables; SLSQP has $O(d^3)$ time complexity; computing the null space is roughly within $O(d^3)$ time; for GP prediction our algorithm only cares about the mean value without the standard deviation, hence it takes $O(Nn)$ time to complete, where n is the number of grid samples. After all generations are calculated, the graph-cuts [7] is performed to reach a continuous Pareto front representation. Representing the number of generation and population sizes as N_{gen} and N_{pop} respectively, the graph-cuts algorithm has $O(N_{gen}N_{pop})$ time complexity if we consider the number of algorithm iterations as a small constant, where $N_{gen}N_{pop}$ equals the maximum number of labels in the graph. As a result, the overall time complexity in this stage is roughly $O(N_{gen}N_{pop}(d^3 + Nn))$.

Batch selection algorithm Given the batch size b, the maximum *performance buffer* capacity D (a number of cells times a maximum number of samples in each cell), the number of points on the current Pareto front N_{pf}, and the number of objectives m, our batch selection algorithm has time complexity $O(bDN_{pf})$ for $m = 2$, $O(bDN_{pf} \log N_{pf})$ for $m = 3$, and $O(bDN_{pf}^{\lfloor (m-1)/2 \rfloor + 1})$ for $m > 3$ due to the algorithm complexity difference for hypervolume calculation w.r.t. m.

B.2 Algorithm runtime comparison

To empirically investigate the efficiency of different algorithms, we record the runtime statistics in seconds as shown in Table B.1. This comparison is done with batch size as 10 and 20 algorithm iterations, and the statistics are averaged across 6 different random seeds and all the iterations. Note that this comparison is highly implementation-dependent and is done by using our codebase. We run this comparison on an Intel Xeon(R) W-2195 CPU @ 2.30GHz * 36 processor. For ParEGO and DGEMO they can be easily parallelized, hence we record the runtime of experiments using 36 parallel processes. For NSGA-II, MOEA/D-EGO, TSEMO, and USeMO-EI, the serial runtime is recorded. From this table, we can see that NSGA-II is the most time-efficient algorithm since it operates simply without Bayesian optimization. For all the problems

Table B.1: Algorithm runtime comparison (seconds per iteration).

Problem	NSGA-II	ParEGO	MOEA/D-EGO	TSEMO	USeMO-EI	DGEMO (Ours)
ZDT1	0.01	16.39	25.39	1.90	1.82	5.51
ZDT2	0.01	13.34	25.09	1.89	1.80	5.24
ZDT3	0.01	15.40	25.24	1.73	1.84	5.45
DTLZ1	0.01	11.94	25.39	1.82	2.26	6.53
DTLZ2	0.01	14.05	25.14	1.50	1.78	4.76
DTLZ3	0.01	11.85	25.34	1.48	1.88	6.84
DTLZ4	0.01	13.76	25.46	1.65	1.93	7.33
DTLZ5	0.01	13.79	25.37	1.51	1.78	4.72
DTLZ6	0.01	10.57	24.75	1.62	1.56	5.69
OKA1	0.01	6.10	24.35	1.52	1.50	4.97
OKA2	0.01	9.28	24.59	1.52	1.60	5.78
VLMOP2	0.01	9.08	25.07	1.47	1.78	5.11
VLMOP3	0.01	7.10	29.24	2.18	1.90	6.38
RE1	0.01	14.37	24.85	1.99	1.66	4.60
RE2	0.01	7.03	24.66	1.61	1.65	4.44
RE3	0.01	7.90	24.56	1.65	1.65	4.77
RE4	0.01	12.60	28.79	2.02	2.25	8.12
RE5	0.01	17.92	28.91	2.89	1.99	7.00
RE6	0.01	6.74	28.89	1.81	1.95	7.03
RE7	0.01	17.91	28.86	4.82	1.98	6.66

we test they have an analytical form of evaluation function thus the evaluation could be done extremely fast. However, in practice, many real-world problems require physical experiments for evaluation which could often take hours to days to finish. Hence for these real-world problems, the runtime cost of the algorithm would be negligible (commonly less than the 30s per iteration).

Appendix C

Ablation Studies

C.1 Batch size

We conduct experiments using different batch sizes, including 1 (Figure C-1), 2 (Figure C-2), 4 (Figure C-3), 5 (Figure C-4), and 20 (Figure C-5) and compare performance based on the hypervolume indicator. The result shows our DGEMO still consistently outperforms other algorithms and has a robust performance over a wide range of batch size.

C.2 Pareto front approximation

To investigate whether the piecewise continuous Pareto front approximation obtained from Section 4.2 helps improve the performance, we do another set of ablation experiments. We remove the last part of the Pareto front approximation algorithm with KKT conditions, and only get the solutions obtained after the local optimization step (see Section 4.2). From these solutions we select the batch of samples to evaluate according to the maximal hypervolume improvement criterion, without the diversity metric. The comparison result is shown in Figure C-6, which demonstrates that DGEMO highly benefits from the continuous Pareto front approximation, as denser candidate solutions are provided and the diversity information becomes available from this representation.

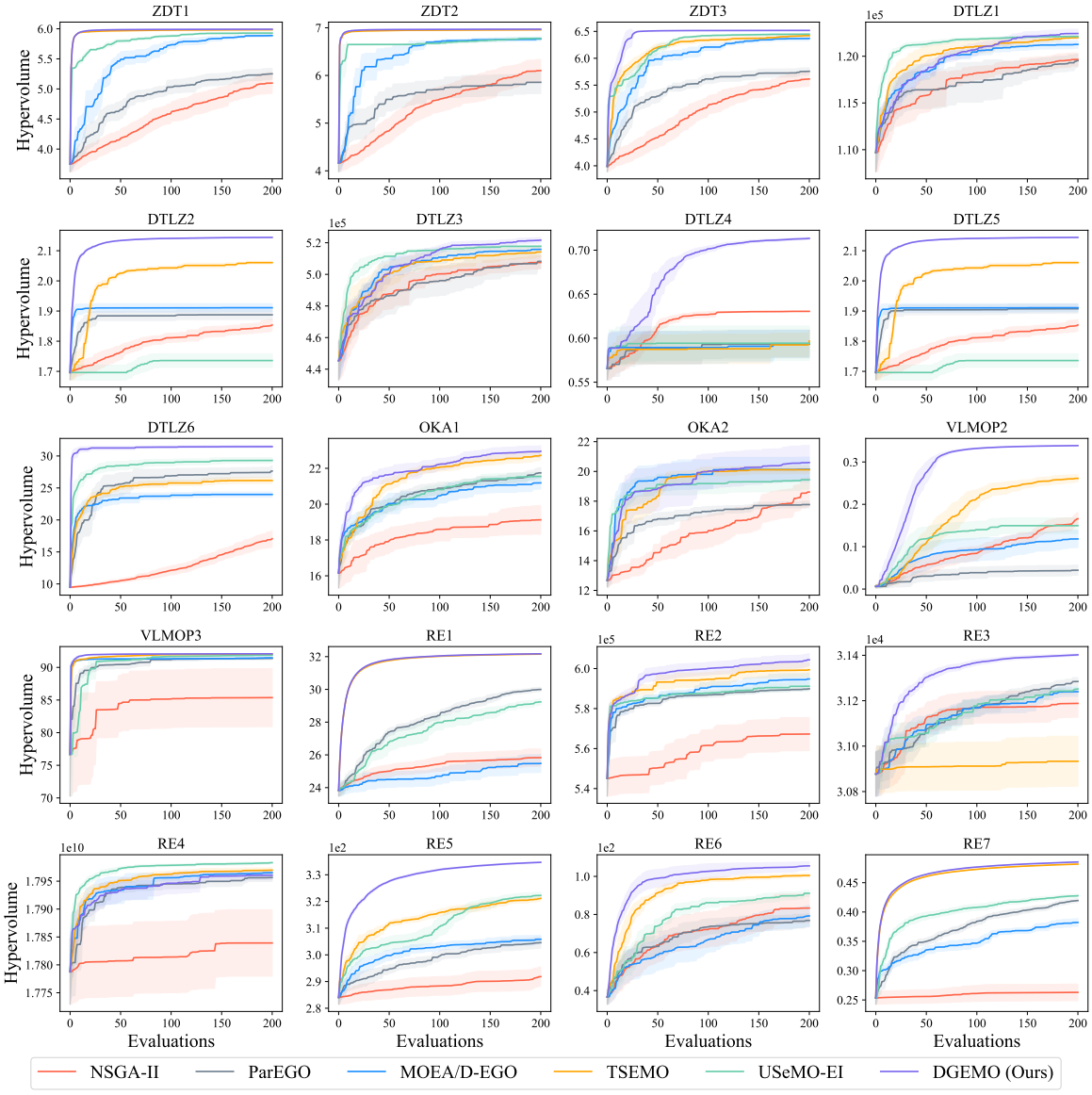


Figure C-1: Hypervolume indicator of different algorithms on all problems with batch size as 1, shown with respect to the different number of function evaluations.

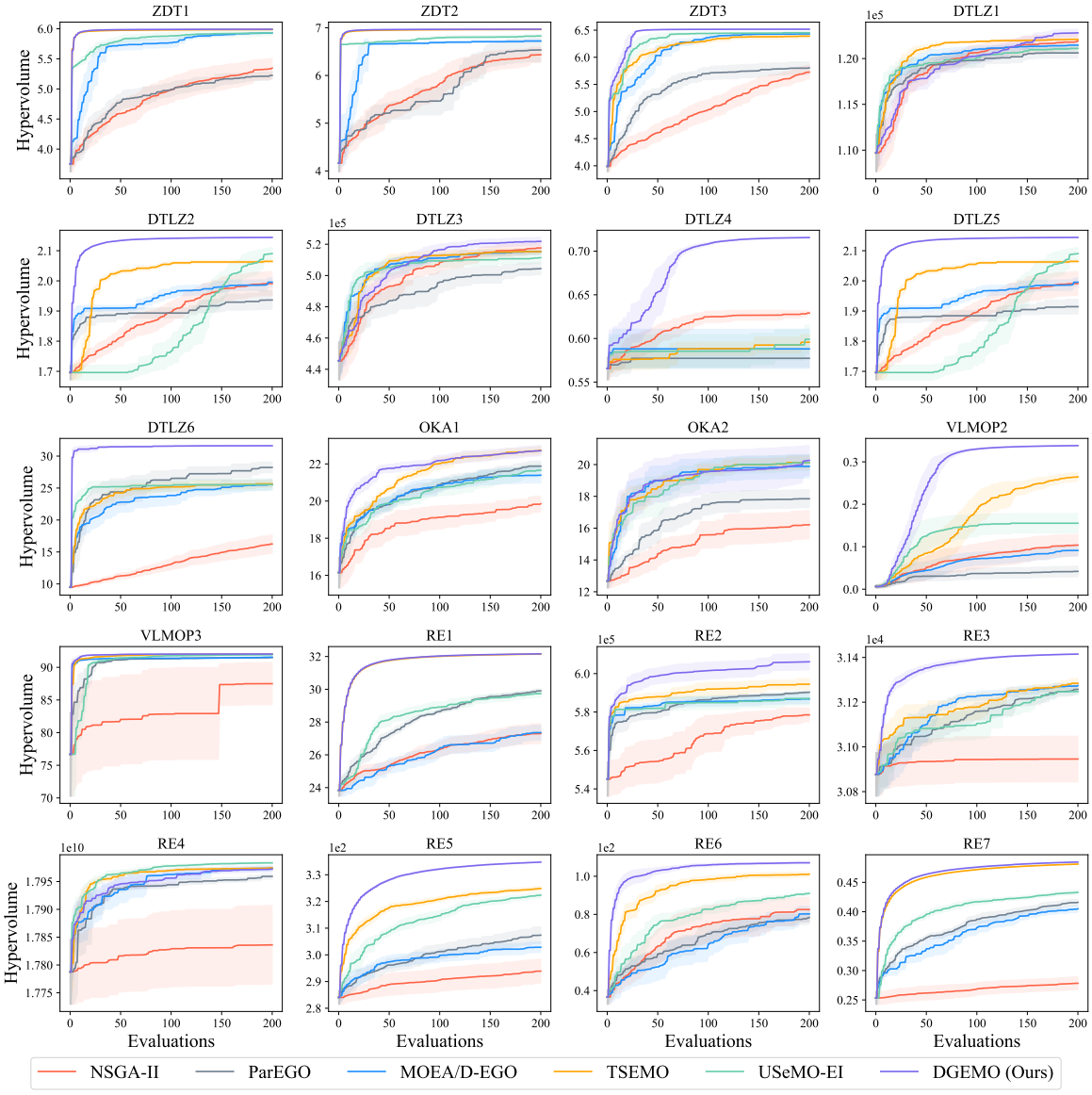


Figure C-2: Hypervolume indicator of different algorithms on all problems with batch size as 2, shown with respect to the different number of function evaluations.

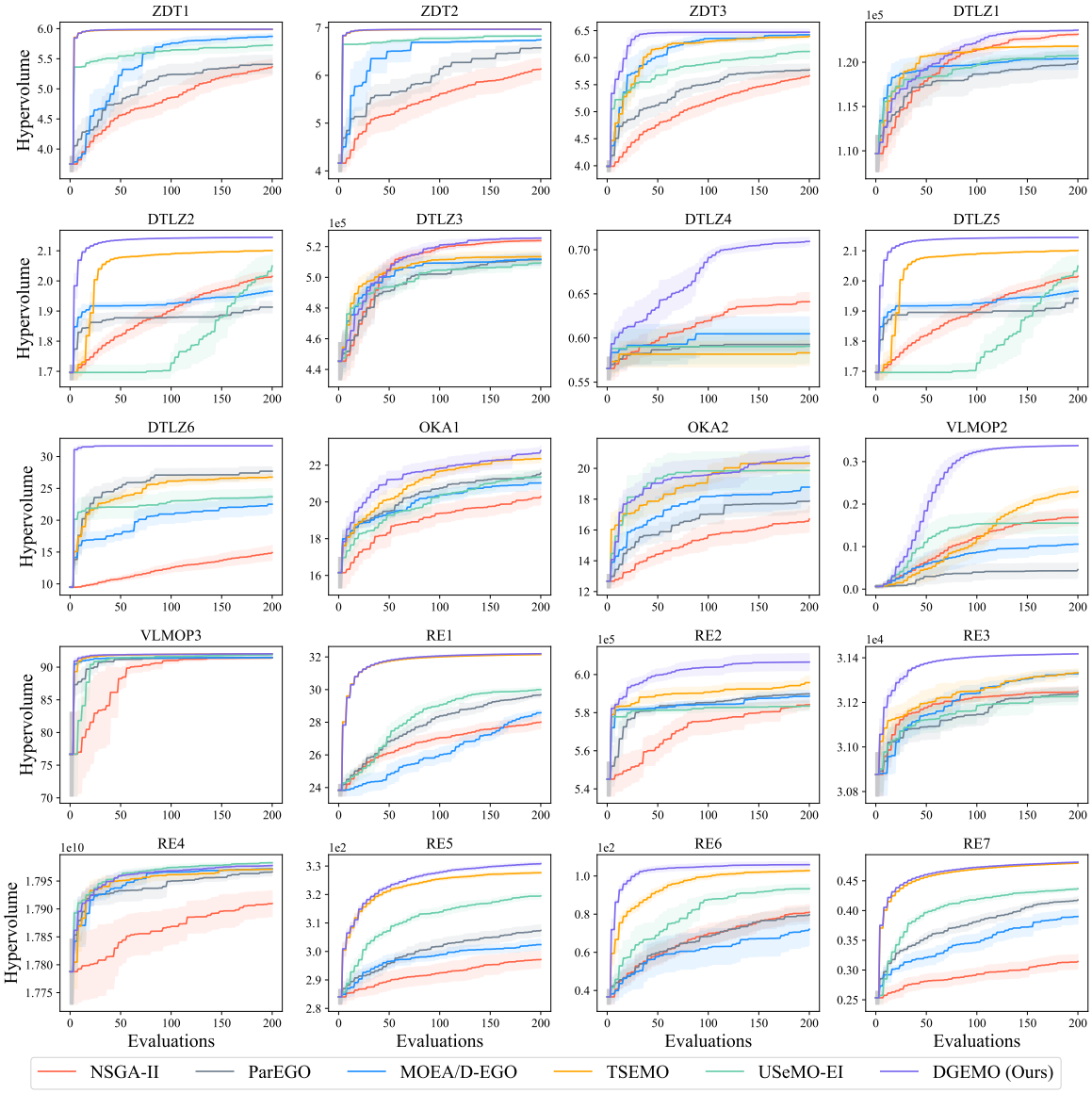


Figure C-3: Hypervolume indicator of different algorithms on all problems with batch size as 4, shown with respect to the different number of function evaluations.

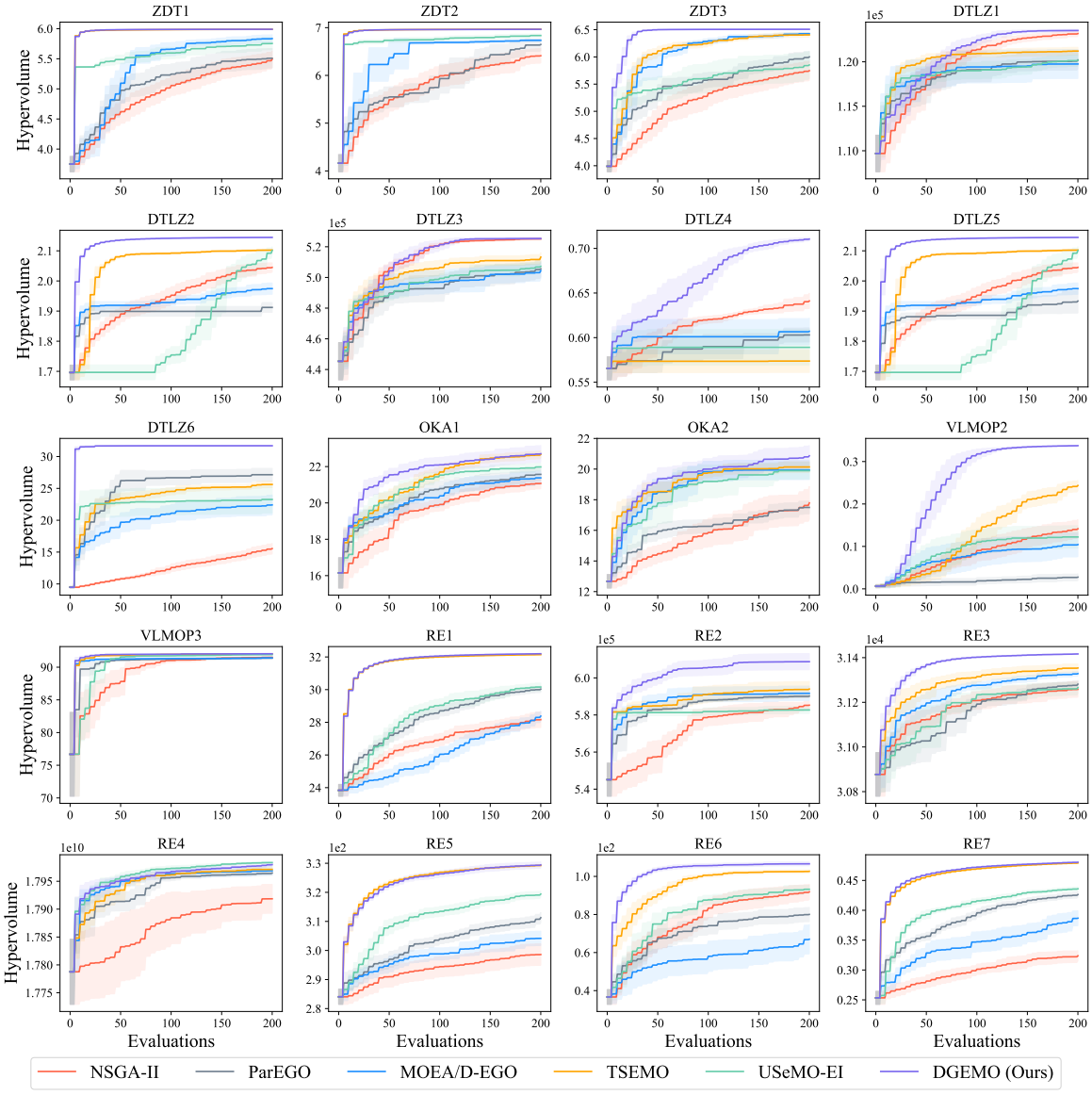


Figure C-4: Hypervolume indicator of different algorithms on all problems with batch size as 5, shown with respect to the different number of function evaluations.

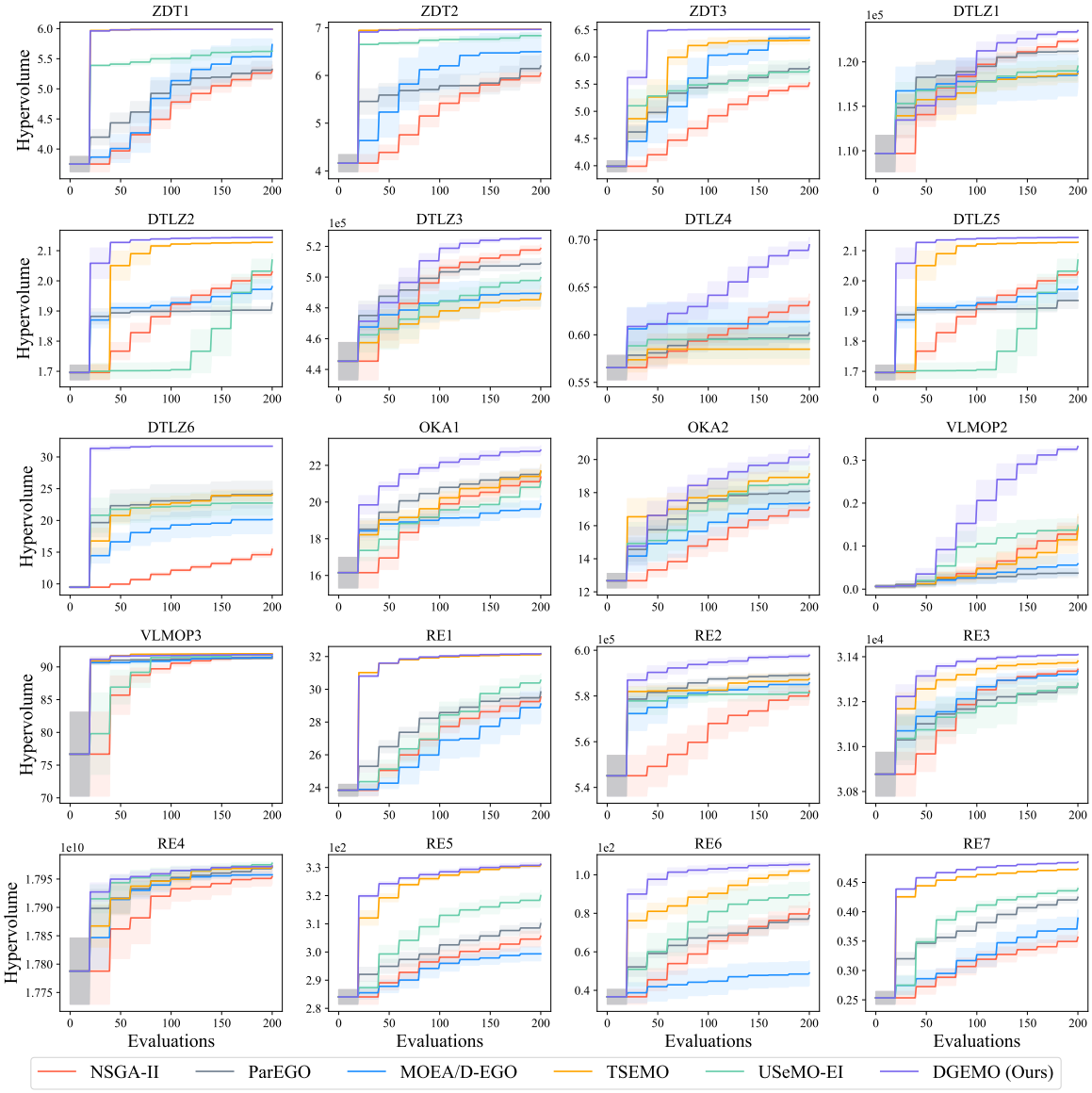


Figure C-5: Hypervolume indicator of different algorithms on all problems with batch size as 20, shown with respect to the different number of function evaluations.

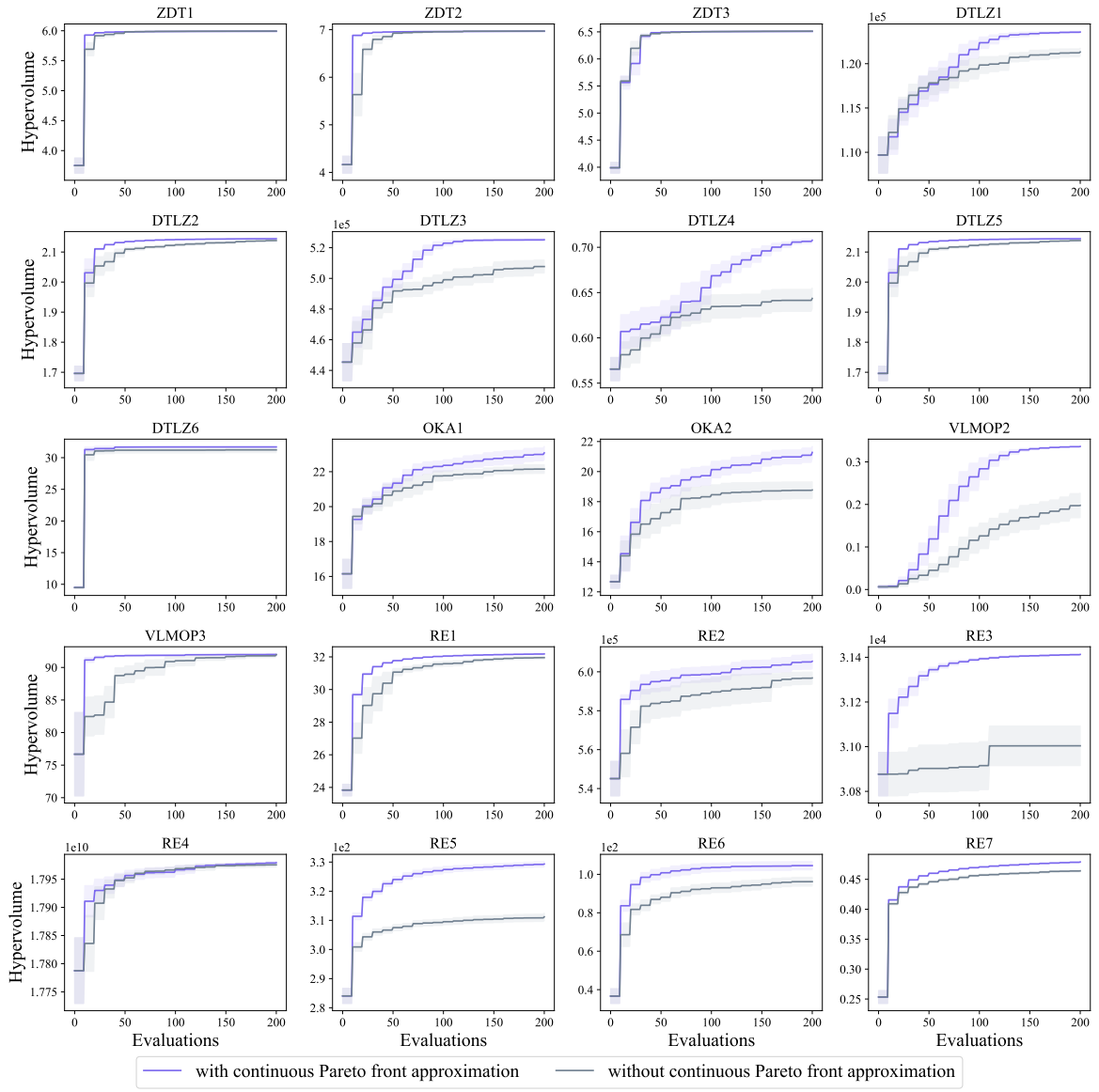


Figure C-6: Hypervolume indicator of DGEMO and its version without continuous Pareto front approximation on all problems with batch size as 10, shown with respect to a different number of function evaluations. Note that the version without the continuous Pareto front approximation does not have the diversity metric incorporated in the selection strategy, but only uses the maximal hypervolume improvement information.

Bibliography

[1] Hossain M Amir and Takashi Hasegawa. Nonlinear mixed-discrete structural optimization. *Journal of Structural Engineering*, 115(3):626–646, 1989.

[2] Peter M Attia, Aditya Grover, Norman Jin, Kristen A Severson, Todor M Markov, Yang-Hung Liao, Michael H Chen, Bryan Cheong, Nicholas Perkins, Zi Yang, et al. Closed-loop optimization of fast-charging protocols for batteries with machine learning. *Nature*, 578(7795):397–402, 2020.

[3] Maximilian Balandat, Brian Karrer, Daniel R. Jiang, Samuel Daulton, Benjamin Letham, Andrew Gordon Wilson, and Eytan Bakshy. BoTorch: A Framework for Efficient Monte-Carlo Bayesian Optimization. In *Advances in Neural Information Processing Systems 33*, 2020.

[4] Syrine Belakaria and Aryan Deshwal. Uncertainty-aware search framework for multi-objective bayesian optimization. In *AAAI Conference on Artificial Intelligence (AAAI)*, 2020.

[5] Syrine Belakaria, Aryan Deshwal, and Janardhan Rao Doppa. Max-value entropy search for multi-objective bayesian optimization. In *Advances in Neural Information Processing Systems*, pages 7823–7833, 2019.

[6] J. Blank and K. Deb. pymoo: Multi-objective optimization in python. *IEEE Access*, pages 1–1, 2020.

[7] Yuri Boykov, Olga Veksler, and Ramin Zabih. Fast approximate energy minimization via graph cuts. *IEEE Transactions on pattern analysis and machine intelligence*, 23(11):1222–1239, 2001.

[8] Eric Bradford, Artur M Schweidtmann, and Alexei Lapkin. Efficient multiobjective optimization employing gaussian processes, spectral sampling and a genetic algorithm. *Journal of global optimization*, 71(2):407–438, 2018.

[9] FY Cheng and XS Li. Generalized center method for multiobjective engineering optimization. *Engineering Optimization*, 31(5):641–661, 1999.

[10] Carlos A Coello Coello and Margarita Reyes Sierra. A study of the parallelization of a coevolutionary multi-objective evolutionary algorithm. In *Mexican international conference on artificial intelligence*, pages 688–697. Springer, 2004.

[11] Emile Contal, David Buffoni, Alexandre Robicquet, and Nicolas Vayatis. Parallel gaussian process optimization with upper confidence bound and pure exploration. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 225–240. Springer, 2013.

[12] Kalyanmoy Deb, Ram Bhushan Agrawal, et al. Simulated binary crossover for continuous search space.

[13] Kalyanmoy Deb and Mayank Goyal. A combined genetic adaptive search (geneas) for engineering design. 1996.

[14] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and TAMT Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE transactions on evolutionary computation*, 6(2):182–197, 2002.

[15] Kalyanmoy Deb and Aravind Srinivasan. Innovization: Innovating design principles through optimization. In *Proceedings of the 8th annual conference on Genetic and evolutionary computation*, pages 1629–1636, 2006.

[16] Kalyanmoy Deb, Lothar Thiele, Marco Laumanns, and Eckart Zitzler. Scalable test problems for evolutionary multiobjective optimization. In *Evolutionary multiobjective optimization*, pages 105–145. Springer, 2005.

[17] Thomas Desautels, Andreas Krause, and Joel W Burdick. Parallelizing exploration-exploitation tradeoffs in gaussian process bandit optimization. *Journal of Machine Learning Research*, 15:3873–3923, 2014.

[18] Michael Emmerich. The computation of the expected improvement in dominated hypervolume of pareto front approximations.

[19] Adrian Gambier and Essameddin Badreddin. Multi-objective optimal control: An overview. In *2007 IEEE International Conference on Control Applications*, pages 170–175. IEEE, 2007.

[20] Tushar Goel, Rajkumar Vaidyanathan, Raphael T Haftka, Wei Shyy, Nestor V Queipo, and Kevin Tucker. Response surface approximation of pareto optimal front in multi-objective optimization. *Computer methods in applied mechanics and engineering*, 196(4-6):879–893, 2007.

[21] Javier González, Zhenwen Dai, Philipp Hennig, and Neil Lawrence. Batch bayesian optimization via local penalization. In *Artificial intelligence and statistics*, pages 648–657, 2016.

[22] Ryan-Rhys Griffiths and José Miguel Hernández-Lobato. Constrained bayesian optimization for automatic chemical design. *arXiv preprint arXiv:1709.05501*, 2017.

[23] Nikolaus Hansen. The cma evolution strategy: A tutorial. *arXiv preprint arXiv:1604.00772*, 2016.

[24] José Miguel Hernández-Lobato, Matthew W Hoffman, and Zoubin Ghahramani. Predictive entropy search for efficient global optimization of black-box functions. In *Advances in neural information processing systems*, pages 918–926, 2014.

[25] Claus Hillermeier. Generalized homotopy approach to multiobjective optimization. *Journal of Optimization Theory and Applications*, 110:557–583, 2001.

[26] Donald R Jones, Matthias Schonlau, and William J Welch. Efficient global optimization of expensive black-box functions. *Journal of Global optimization*, 13(4):455–492, 1998.

[27] Tarun Kathuria, Amit Deshpande, and Pushmeet Kohli. Batched gaussian process bandit optimization via determinantal point processes. In *Advances in Neural Information Processing Systems*, pages 4206–4214, 2016.

[28] Joshua Knowles. Parego: a hybrid algorithm with on-line landscape approximation for expensive multiobjective optimization problems. *IEEE Transactions on Evolutionary Computation*, 10(1):50–66, 2006.

[29] Dieter Kraft. A software package for sequential quadratic programming. *Forschungsbericht- Deutsche Forschungs- und Versuchsanstalt fur Luft- und Raumfahrt*, 1988.

[30] Harold J Kushner. A new method of locating the maximum point of an arbitrary multipeak curve in the presence of noise. 1964.

[31] Xi Lin, Qingfu Zhang, and Sam Kwong. An efficient batch expensive multiobjective evolutionary algorithm based on decomposition. In *2017 IEEE Congress on Evolutionary Computation (CEC)*, pages 1343–1349. IEEE, 2017.

[32] Xi Lin, Hui-Ling Zhen, Zhenhua Li, Qingfu Zhang, and Sam Kwong. A batched scalable multi-objective bayesian optimization algorithm. *arXiv preprint arXiv:1811.01323*, 2018.

[33] Dong C Liu and Jorge Nocedal. On the limited memory bfgs method for large scale optimization. *Mathematical programming*, 45(1-3):503–528, 1989.

[34] Mina Konaković Luković, Yunsheng Tian, and Wojciech Matusik. Diversity-guided multi-objective bayesian optimization with batch evaluations. In *Advances in Neural Information Processing Systems 33, NeurIPS 2020, December 6-12, 2020, virtual*, 2020.

[35] Roman Marchant and Fabio Ramos. Bayesian optimisation for intelligent environmental monitoring. pages 2242–2249, 10 2012.

[36] Ruben Martinez-Cantin, Nando Freitas, Eric Brochu, Jose Castellanos, and Arnaud Doucet. A bayesian exploration-exploitation approach for optimal online sensing and planning with a visually guided mobile robot. *Auton. Robots*, 27:93–103, 08 2009.

[37] Michael D McKay, Richard J Beckman, and William J Conover. Comparison of three methods for selecting values of input variables in the analysis of output from a computer code. *Technometrics*, 21(2):239–245, 1979.

[38] Kaisa Miettinen. *Nonlinear multiobjective optimization*, volume 12. Springer Science & Business Media, 2012.

[39] Jonas Močkus. On bayesian methods for seeking the extremum. In *Optimization techniques IFIP technical conference*, pages 400–404. Springer, 1975.

[40] Tatsuya Okabe, Yaochu Jin, Markus Olhofer, and Bernhard Sendhoff. On test functions for evolutionary multi-objective optimization. In *International Conference on Parallel Problem Solving from Nature*, pages 792–802. Springer, 2004.

[41] Travis E Oliphant. *A guide to NumPy*, volume 1. Trelgol Publishing USA, 2006.

[42] Victor Picheny. Multiobjective optimization using gaussian process emulators via stepwise uncertainty reduction. *Statistics and Computing*, 25(6):1265–1280, 2015.

[43] Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning)*. The MIT Press, 2005.

[44] Tapabrata Ray and KM Liew. A swarm metaphor for multiobjective design optimization. *Engineering optimization*, 34(2):141–153, 2002.

[45] Nery Riquelme, Christian Von Lücken, and Benjamin Baran. Performance metrics in multi-objective optimization. In *2015 Latin American Computing Conference (CLEI)*, pages 1–11. IEEE, 2015.

[46] Daniel Russo and Benjamin Van Roy. Learning to optimize via information-directed sampling. In *Advances in Neural Information Processing Systems*, pages 1583–1591, 2014.

[47] Adriana Schulz, Harrison Wang, Eitan Grinspun, Justin Solomon, and Wojciech Matusik. Interactive exploration of design trade-offs. *ACM Transactions on Graphics (TOG)*, 37(4):1–14, 2018.

[48] B. Shahriari, K. Swersky, Z. Wang, R. P. Adams, and N. de Freitas. Taking the human out of the loop: A review of bayesian optimization. *Proceedings of the IEEE*, 104(1):148–175, 2016.

[49] Benjamin J Shields, Jason Stevens, Jun Li, Marvin Parasram, Farhan Damani, Jesus I Martinez Alvarado, Jacob M Janey, Ryan P Adams, and Abigail G Doyle. Bayesian reaction optimization as a tool for chemical synthesis. *Nature*, 590(7844):89–96, 2021.

[50] Niranjan Srinivas, Andreas Krause, Sham Kakade, and Matthias Seeger. Gaussian process optimization in the bandit setting: no regret and experimental design. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, pages 1015–1022, 2010.

[51] Raj Subbu, Piero P Bonissone, Neil Eklund, Srinivas Bollapragada, and Kete Chalermkraivuth. Multiobjective financial portfolio design: A hybrid evolutionary approach. In *2005 IEEE Congress on Evolutionary Computation*, volume 2, pages 1722–1729. IEEE, 2005.

[52] Ryoji Tanabe and Hisao Ishibuchi. An easy-to-use real-world multi-objective optimization problem suite. *Applied Soft Computing*, page 106078, 2020.

[53] David A Van Veldhuizen and Gary B Lamont. Multiobjective evolutionary algorithm test suites. In *Proceedings of the 1999 ACM symposium on Applied computing*, pages 351–357, 1999.

[54] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, CJ Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake Vand erPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1. 0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 2020.

[55] J. Wu, W.Y. Zhang, S. Zhang, Y.N. Liu, and X.H. Meng. A matrix-based bayesian approach for manufacturing resource allocation planning in supply chain management. *International Journal of Production Research*, 51(5):1451–1463, 2013.

[56] Zhenning Yu, Viswanathan Ramakrishnan, and Caitlyn Meinzer. Simulation optimization for bayesian multi-arm multi-stage clinical trial with binary endpoints. *Journal of Biopharmaceutical Statistics*, 29:1–12, 02 2019.

[57] Q. Zhang and H. Li. Moea/d: A multiobjective evolutionary algorithm based on decomposition. *IEEE Transactions on Evolutionary Computation*, 11(6):712–731, 2007.

[58] Qingfu Zhang, Wudong Liu, Edward Tsang, and Botond Virginas. Expensive multiobjective optimization by moea/d with gaussian process model. *IEEE Transactions on Evolutionary Computation*, 14(3):456–474, 2009.

[59] Yichi Zhang, Daniel W Apley, and Wei Chen. Bayesian optimization for materials design with mixed quantitative and qualitative variables. *Scientific Reports*, 10(1):1–13, 2020.

[60] Eckart Zitzler, Kalyanmoy Deb, and Lothar Thiele. Comparison of multiobjective evolutionary algorithms: Empirical results. *Evolutionary computation*, 8(2):173–195, 2000.

[61] Eckart Zitzler and Lothar Thiele. Multiobjective evolutionary algorithms: a comparative case study and the strength pareto approach. *IEEE transactions on Evolutionary Computation*, 3(4):257–271, 1999.

[62] Marcela Zuluaga, Guillaume Sergent, Andreas Krause, and Markus Püschel. Active learning for multi-objective optimization. In *International Conference on Machine Learning*, pages 462–470, 2013.