Journal of Artificial Intelligence Research 73 (2022) 1209-1250 Submitted 07/2021; published 04/2022

Multiobjective Tree-Structured Parzen Estimator

Yoshihiko Ozaki OZAKI-Y@AIST.GO.JP
Artificial Intelligence Research Center, AIST, Tokyo, Japan
GREE, Inc., Tokyo, Japan

Yuki Tanigaki TANIGAKI.YUKI@AIST.GO.JP
Artificial Intelligence Research Center, AIST, Tokyo, Japan

Shuhei Watanabe WATANABS@CS.UNI-FREIBURG.DE
University of Freiburg, Freiburg, Germany

Masahiro Nomura NOMURA_MASAHIRO@CYBERAGENT.CO.JP
CyberAgent, Inc., Tokyo, Japan

Masaki Onishi ONISHI@NI.AIST.GO.JP
Artificial Intelligence Research Center, AIST, Tokyo, Japan

Abstract

Practitioners often encounter challenging real-world problems that involve a simultaneous optimization of multiple objectives in a complex search space. To address these problems, we propose a practical multiobjective Bayesian optimization algorithm. It is an extension of the widely used Tree-structured Parzen Estimator (TPE) algorithm, called Multiobjective Tree-structured Parzen Estimator (MOTPE). We demonstrate that MOTPE approximates the Pareto fronts of a variety of benchmark problems and a convolutional neural network design problem better than existing methods through the numerical results. We also investigate how the configuration of MOTPE affects the behavior and the performance of the method and the effectiveness of asynchronous parallelization of the method based on the empirical results.

1. Introduction

Many real-world problems get involved in optimizing multiple objectives simultaneously in a complex search space. These objectives are often conflicting with one another, computationally and/or financially expensive to evaluate, and blackbox, i.e., their analytical forms are not available. One of such problems is the design of mechanical products such as a diesel engine combustion chamber (Jeong, Minemura, & Obayashi, 2006) and a tractor's air intake ventilation system (Chugh, Sindhya, Miettinen, Jin, Kratky, & Makkonen, 2017). These mechanical design problems usually require expensive computation (e.g., fluid dynamics simulations) and financially expensive prototyping processes. Another example is an automated machine learning problem to find a preferable model in terms of multiple criteria such as high prediction accuracy and low computational cost. This problem is known as Hyperparameter Optimization (HPO) (Igel, 2005; Shah & Ghahramani, 2016; Hernández-Lobato, Hernández-Lobato, Shah, & Adams, 2016; Horn & Bischl, 2016; Belakaria, Deshwal, & Doppa, 2019) or Neural Architecture Search (NAS) (Lu, Whalen, Boddeti, Dhebar, Deb, Goodman, & Banzhaf, 2019). In this problem, the objectives require time-consuming training of the machine learning model to evaluate. The search space can be complex because

©2022 AI Access Foundation. All rights reserved.

some typical hyperparameters, e.g., the number of layers and the type of activation function, are non-continuous. Moreover, if we use cloud computing resources, the cloud usage fee will be also charged, i.e., this problem can be financially expensive.

Objective functions of real-world problems, such as those mentioned above, are expensive to evaluate. Therefore, such a real-world application needs an optimization algorithm that works with a limited evaluation budget and/or can be effectively parallelized. One of the most successful approaches to tackle expensive optimization problems is utilizing a surrogate that approximates an objective landscape or a promising region in a search space for efficient search. Such surrogate-based algorithms for single- and multi-objective optimization have been widely studied in the machine learning and the evolutionary computation community as Bayesian optimization (Archetti & Candelieri, 2019) and surrogate-assisted evolutionary algorithm (Chugh, Sindhya, Hakanen, & Miettinen, 2019). Currently, the Gaussian process (GP, also called Kriging) (Rasmussen & Williams, 2006) is the de facto standard model in surrogate-based algorithms. Most surrogate-based optimization algorithms successful in real-world problems adopt GP as a surrogate model, e.g., PESMO (Hernández-Lobato et al., 2016) and K-RVEA (Chugh et al., 2017; Chugh, Jin, Miettinen, Hakanen, & Sindhya, 2018). GP predicts expected values and uncertainties of objectives that are essential for balancing between exploration and exploitation during optimization. However, there are well-known drawbacks of standard GP: it is not so suitable for non-continuous search space; it suffers from high computational complexity to data size. These are problematic because real-world problems often have complex search space, and low scalability severely constrains our evaluation budget even when parallel evaluation is allowed. To address them, we generally have two approaches: introducing advanced techniques into GP to improve its performance; employing a surrogate other than GP. As for the first approach, a number of handling and approximation techniques have been proposed (Qian, Wu, & Wu, 2008; Liu, Ong, Shen, & Cai, 2020; Zhang, Apley, & Chen, 2020; Cuesta-Ramirez, Riche, Roustant, Perrin, Durantin, & Gliere, 2021; Pelamatti, Brevault, Balesdent, Talbi, & Guerin, 2021). These techniques generally help practitioners apply GP-based optimization algorithms to a variety of real-world problems. However, each technique often has its drawbacks, e.g., approximation causes prediction performance degradation. As for the second approach, non-GP-based optimization algorithms have been proposed (Bergstra, Bardenet, Bengio, & Kégl, 2011; Hutter, Hoos, & Leyton-Brown, 2011). One of the most notable algorithms is Tree-structured Parzen Estimator (TPE) (Bergstra et al., 2011; Bergstra, Yamins, & Cox, 2013), which is known for the standard solver of Hyperopt (Bergstra, Komer, Eliasmith, Yamins, & Cox, 2015) and Optuna (Akiba, Sano, Yanase, Ohta, & Koyama, 2019) – open-source software for HPO that are widely used in the machine learning community. TPE is suitable for computationally expensive problems because of its efficient search with a surrogate model and parallelizability. Its surrogate based on the Parzen estimators (Parzen, 1962) can naturally handle complex search space, scale to tens of variables, and scale to at least one thousand observations. These properties make TPE better than GP-based optimization algorithms in certain types of problems. Bergstra et al. (2011) reported that TPE outperformed a GP-based Bayesian optimization algorithm in single-objective HPO including non-continuous variables. However, unfortunately, TPE is not designed for multi-objective optimization, so its application is limited to single-objective problems. After all, a more powerful algorithm is desired by practitioners.

Motivated by the above discussion, in this study, we develop an optimization algorithm that can handle multiple objectives and complex search space with scalability and parallelizability working with a limited budget by extending TPE. We address the following two nontrivial issues to extend TPE: (1) designing a proper strategy to split observations including many incomparable ones and (2) deriving an effective acquisition function to handle multiple objectives. This paper covers the extension of TPE to multiobjective optimization in detail and demonstrates the performance of the proposed method through several numerical results including a real-world problem.

1.1 Contributions

The contributions of this study are summarized below.

- We propose a practical multiobjective Bayesian optimization algorithm called Multiobjective Tree-structured Parzen Estimator (MOTPE), which is an extension of TPE.
- We compare the proposed MOTPE and other surrogate-based multiobjective optimization algorithms on benchmark problems. The empirical results demonstrate that, in most cases, MOTPE approximates the Pareto front better than the other algorithms with a limited evaluation budget.
- We compare MOTPE and NSGA-II (Deb, Pratap, Agarwal, & Meyarivan, 2002) on medium-dimensional benchmark problems under a medium budget setting. The empirical results give us insight into when we should use MOTPE rather than NSGA-II.
- We solve a multiobjective convolutional neural network (CNN) design problem, which is an important real-world problem, using MOTPE and existing algorithms. Our results show that MOTPE handles the complex search space better than the other algorithms.
- We discuss how the configuration of MOTPE affects both its behavior and performance through experiments. We also provide an empirical recommendation of the configuration based on the numerical results.
- We demonstrate the effectiveness of parallelization in MOTPE. Our results show that asynchronous parallelization drastically speeds up the method.

The preliminary version of this paper appeared in Proceedings of the 2020 Genetic and Evolutionary Computation Conference (GECCO). The major updates from the last version (Ozaki, Tanigaki, Watanabe, & Onishi, 2020) are the following.

- We stabilize MOTPE by introducing a new weighting strategy that assigns weights to observations proportional to hypervolume contributions.
- We introduce the parallelization of MOTPE that is vital for solving real-world problems in practice.
- We update the numerical results section entirely. Especially, we demonstrate that MOTPE has the practicality sufficiently to solve a multiobjective CNN design problem.

2. Background

This section first provides definitions and notations that we use. Then, the problem addressed in this paper is described. After that, the TPE algorithm for single-objective optimization is explained.

2.1 Mathematical Preliminaries

The following definitions and notations are used in this paper.

Definition 1 (Dominance relation)**.** A vector $\boldsymbol{y} \in \mathbb{R}^m$ dominates a vector $\boldsymbol{y}' \in \mathbb{R}^m$ iff $\forall i : y_i \leq y'_i$ and $\exists i : y_i < y'_i$, denoted $\boldsymbol{y} \prec \boldsymbol{y}'$. A vector $\boldsymbol{y} \in \mathbb{R}^m$ weakly dominates a vector $\boldsymbol{y}' \in \mathbb{R}^m$ iff $\forall i : y_i \leq y'_i$, denoted $\boldsymbol{y} \preceq \boldsymbol{y}'$. (Here we assume that smaller y_i is more preferable, cf. Section 2.2.)

Definition 2 (Incomparable relation)**.** Two vectors $\boldsymbol{y} \in \mathbb{R}^m$ and $\boldsymbol{y}' \in \mathbb{R}^m$ are incomparable iff neither $\boldsymbol{y} \preceq \boldsymbol{y}'$ nor $\boldsymbol{y}' \preceq \boldsymbol{y}$, denoted $\boldsymbol{y} \parallel \boldsymbol{y}'$.

Definition 3 (Nondomination rank)**.** For a finite set of vectors $Y \subset \mathbb{R}^m$, the nondomination rank (Deb et al., 2002) of a vector $\boldsymbol{y} \in Y$ denoted $\mathrm{rank}(\boldsymbol{y}) \in \mathbb{N}$ is defined as follows:

- $\mathrm{rank}(\boldsymbol{y}) = 1$ iff $\nexists \boldsymbol{y}' \in Y : \boldsymbol{y}' \prec \boldsymbol{y}$.
- $\mathrm{rank}(\boldsymbol{y}) = \max_{\boldsymbol{y}' \in Y'} \mathrm{rank}(\boldsymbol{y}') + 1$ where $Y' = \{\boldsymbol{y}' \in Y \mid \boldsymbol{y}' \prec \boldsymbol{y}\}$.

A vector $\boldsymbol{y} \in Y$ is nondominated iff $\mathrm{rank}(\boldsymbol{y}) = 1$. We denote by $Y_{\mathrm{rank}(k)}$ the set $\{\boldsymbol{y} \in Y \mid \mathrm{rank}(\boldsymbol{y}) = k\}$.

Definition 4 (Dominance relation between set and vector)**.** For a finite set of vectors $Y \subset \mathbb{R}^m$ and a vector $\boldsymbol{y} \in \mathbb{R}^m$, define $Y \prec \boldsymbol{y}$ (resp. $Y \preceq \boldsymbol{y}$) iff $\exists \boldsymbol{y}' \in Y_{\mathrm{rank}(1)} : \boldsymbol{y}' \prec \boldsymbol{y}$ (resp. $\boldsymbol{y}' \preceq \boldsymbol{y}$). For a finite set of vectors $Y \subset \mathbb{R}^m$ and a vector $\boldsymbol{y} \in \mathbb{R}^m$, also define $\boldsymbol{y} \prec Y$ (resp. $\boldsymbol{y} \preceq Y$) iff $\exists \boldsymbol{y}' \in Y_{\mathrm{rank}(1)} : \boldsymbol{y} \prec \boldsymbol{y}'$ (resp. $\boldsymbol{y} \preceq \boldsymbol{y}'$).

Definition 5 (Incomparable relation between set and vector)**.** For a finite set of vectors $Y \subset \mathbb{R}^m$ and a vector $\boldsymbol{y} \in \mathbb{R}^m$, define $Y \parallel \boldsymbol{y}$ (also denoted $\boldsymbol{y} \parallel Y$) iff $\forall \boldsymbol{y}' \in Y_{\mathrm{rank}(1)} : \boldsymbol{y} \parallel \boldsymbol{y}'$.

Definition 6 (Pareto optimality)**.** For a function $\boldsymbol{f} : X \to \mathbb{R}^m$, a vector $\boldsymbol{x} \in X$ is Pareto optimal iff $\nexists \boldsymbol{x}' \in X : \boldsymbol{f}(\boldsymbol{x}') \prec \boldsymbol{f}(\boldsymbol{x})$. An entire set of Pareto optimal vectors $\{\boldsymbol{x} \in X \mid \nexists \boldsymbol{x}' \in X : \boldsymbol{f}(\boldsymbol{x}') \prec \boldsymbol{f}(\boldsymbol{x})\}$ is called the Pareto set. The set of corresponding images to the Pareto set $\{\boldsymbol{f}(\boldsymbol{x}) \in \mathbb{R}^m \mid \boldsymbol{x} \in X \text{ is Pareto optimal}\}$ is called the Pareto front.

Definition 7 (Hypervolume indicator)**.** Let $\lambda(S)$ denote the Lebesgue measure of a measurable set S. The hypervolume indicator I_{H} of a finite set of vectors $Y \subset \mathbb{R}^m$ with a reference point $\boldsymbol{r} \in \mathbb{R}^m$ is defined as:

$$I_{\mathrm{H}}(Y; \boldsymbol{r}) \coloneqq \lambda(\{\boldsymbol{y} \in \mathbb{R}^m \mid Y \preceq \boldsymbol{y} \preceq \boldsymbol{r}\}). \tag{1}$$

Hereafter, we omit $\boldsymbol{r}$ and simply use $I_{\mathrm{H}}(Y)$ when our discussion does not depend on a specific choice of $\boldsymbol{r}$.

Definition 8 (Hypervolume contribution)**.** For a finite set of vectors $Y \subset \mathbb{R}^m$ and a vector $\boldsymbol{y} \in Y$, the hypervolume contribution of $\boldsymbol{y}$ for Y is defined as $I_{\mathrm{H}}(Y) - I_{\mathrm{H}}(Y \setminus \{\boldsymbol{y}\})$.

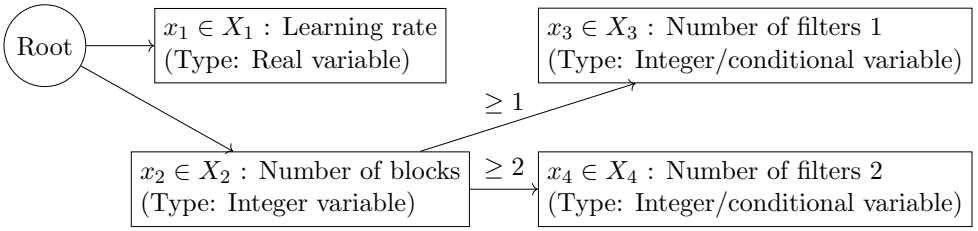


Figure 1: A tree-structured search space of HPO of a neural network with four parameters: Learning rate, Number of blocks, Number of filters 1, and Number of filters 2. x_3 is active only when $x_2 \geq 1$. x_4 is active only when $x_2 \geq 2$.

2.2 Problem Statement

The problem addressed in this paper is formulated as:

$$\begin{aligned} &\text{minimize} \quad \boldsymbol{f}(\boldsymbol{x}) \coloneqq (f_1(\boldsymbol{x}), \ldots, f_m(\boldsymbol{x})) \\ &\boldsymbol{x} \in X \end{aligned} \tag{2}$$

where X is a *tree-structured* search space composed of the search space X_i of each variable (also called parameter) x_i ($i = 1, \ldots, n$), and $f_j : X \to \mathbb{R}$ ($j = 1, \ldots, m$) is an objective function. The objectives are assumed to be conflicting with one another, expensive to evaluate, and blackbox. Without loss of generality, the minimization of all objectives is considered. The search space can be a mixture of real, integer, and categorical variables. This search space is tree-structured in the sense that some child variables are only active when their parent variables take particular values (Bergstra et al., 2011). In this paper, we call such child variables *conditional*. Non-conditional variables are always treated as active. Figure 1 shows a tree-structured search space example of HPO of a neural network with four parameters. The most important property of tree-structured search space is that only active parameters are required to evaluate the objectives, e.g., in the example search space, we can evaluate the objectives without x_4 when $x_2 = 1$. Note that standard hyper-rectangle search space is a special case of tree-structured search space. The goal of this problem is to efficiently approximate the Pareto front with a given evaluation budget.

2.3 Tree-Structured Parzen Estimator

We introduce TPE (Bergstra et al., 2011) for single-objective minimization in Algorithm 1 because TPE is the principal basis of MOTPE.

Let X be a tree-structured search space and $f : X \to \mathbb{R}$ be an objective function. We consider the problem finding $\boldsymbol{x}^\star \in \operatorname{argmin}_{\boldsymbol{x} \in X} f(\boldsymbol{x})$. Assume a set of observations $D = \{(\boldsymbol{x}^{(1)}, y^{(1)}), \ldots, (\boldsymbol{x}^{(k)}, y^{(k)})\}$. TPE models $p(x_i \mid y)$ for each parameter x_i ($\in X_i$) using the two probability density functions as follows:

$$p(x_i \mid y) = \begin{cases} l(x_i) & \text{if } y < y^* \\ g(x_i) & \text{if } y^* \leq y \end{cases} \tag{3}$$

Algorithm 1 Tree-structured Parzen Estimator

Require:
$D = \{(\boldsymbol{x}^{(1)}, y^{(1)}), \ldots, (\boldsymbol{x}^{(k)}, y^{(k)})\}$: observations
$n_{\mathrm{t}} \in \mathbb{N}$: number of iterations
$n_{\mathrm{c}} \in \mathbb{N}$: number of candidates
$\gamma \in (0, 1)$: quantile

1: **for** $t \leftarrow 1, \ldots, n_{\mathrm{t}}$ **do**
2: $D_l \leftarrow \{(\boldsymbol{x}, y) \in D \mid y$ is included in the best-$\lfloor \gamma |D| \rfloor$ objective values in $D\}$
3: $D_g \leftarrow D \setminus D_l$
4: **repeat**
5: $i \leftarrow i \in [1, n]$ such that x_i is active and x_i^* has not been sampled
6: construct $l(x_i)$ with $\{x_i \mid (\boldsymbol{x}, y) \in D_l\}$ and $g(x_i)$ with $\{x_i \mid (\boldsymbol{x}, y) \in D_g\}$
7: $C_i \leftarrow \{x_i^{(j)} \sim l(x_i) \mid j = 1, \ldots, n_{\mathrm{c}}\}$ ▷ sample n_{c} candidates for x_i^*
8: $x_i^* \leftarrow \mathrm{argmax}_{x_i \in C_i} \, l(x_i)/g(x_i)$ ▷ approximate $\mathrm{argmax}_{x_i \in X_i} \, l(x_i)/g(x_i)$
9: **until** all active parameters have been sampled
10: $D \leftarrow D \cup \{(\boldsymbol{x}^*, f(\boldsymbol{x}^*))\}$ ▷ $\boldsymbol{x}^*$ is the vector composed of all sampled x_i^*
11: **end for**
12: **return** $\boldsymbol{x}$ with the minimum y value in D

where $l(x_i)$ is constructed using the subset of the observed x_i values $\{x_i^{(j)} \in X_i \mid y^{(j)} \ (= f(\boldsymbol{x}^{(j)})) < y^*, \ j = 1, \ldots, k\}$, and $g(x_i)$ is constructed using the set of the remaining observed x_i values. The value y^* is selected to be a quantile $\gamma \in (0, 1)$ of the observed y values satisfying $p(y < y^*) = \gamma$. In lines 2–3 of Algorithm 1, the observations are split into D_l and D_g to construct the models. This can be easily achieved by sorting the observations based on the y values. In line 6, observations such that x_i is inactive are ignored. Briefly, $l(x_i)$ models the density of good x_i values whereas $g(x_i)$ models the density of poor x_i values. In TPE, these densities are estimated as described in Section 2.3.1.

2.3.1 Density Estimation

Given a set of observed x_i values $D_x = \{x_i \mid (\boldsymbol{x}, y) \in D_l \text{ (or } D_g)\}$, TPE estimates the density of the observations in a different way based on the type of x_i, i.e., real, integer, or categorical, and the scale of the parameter that we set, i.e., uniform or log-uniform for the real/integer parameter.

For a real/integer parameter x_i with a uniform scale, the following Parzen estimator is used to estimate the density:

$$p(x_i) = \frac{\sum_{x_i' \in D_x} w_{x_i'} k(x_i, x_i') + w_{\mathrm{p}} k(x_i, x_{\mathrm{p}})}{\sum_{x_i' \in D_x} w_{x_i'} + w_{\mathrm{p}}} \tag{4}$$

where $w_{x_i'}$ is a weight for the observation x_i', x_{p} is a fixed prior, w_{p} is a prior weight, and $k : X_i \times X_i \to \mathbb{R}$ is a kernel function such that $\int_x k(x, \cdot) dx = 1$. As a kernel function, TPE uses the truncated Gaussian kernel. In other words, the density estimator is constructed by placing densities in the form of truncated Gaussian distributions $\mathcal{N}_{\mathrm{trunc}}(\mu, \sigma, a, b)$ at each value of x_i' (i.e., $\mu = x_i'$). a and b are the lower bound and the upper bound of the defined

domain, thus $a = \inf X_i$ and $b = \sup X_i$. The bandwidth selection is performed point-wisely as follows:

$$\sigma = \min(\max(x'_i - x_{\mathrm{L}}, x_{\mathrm{R}} - x'_i, \epsilon), b - a) \tag{5}$$

where $x_{\mathrm{L}} = \max(\{x \mid x \in D_x \cup \{a\},\ x < x'_i\})$, $x_{\mathrm{R}} = \min(\{x \mid x \in D_x \cup \{b\},\ x > x'_i\})$, and $\epsilon = (b-a)/\min(100, 1+|D_x \cup \{x_{\mathrm{p}}\}|)$ is a constant that prevents extremely small bandwidth. The fixed prior is set to $x_{\mathrm{p}} = (a+b)/2$, $\sigma = b - a$, and $w_{\mathrm{p}} = 1$. When the parameter scale is set to log-uniform, the parameter is treated as uniform in the log domain.

For a categorical parameter x_i, a simple weighted histogram is used to estimate the density. In this case, the probability of each choice c in X_i is chosen proportionally to $\sum_{x'_i \in D_x} w_{x'_i} \mathbf{1}_c(x'_i) + w_{\mathrm{p}}$ where

$$\mathbf{1}_c(x) = \begin{cases} 1 & \text{if } x = c \\ 0 & \text{otherwise} \end{cases} \tag{6}$$

and $w_{\mathrm{p}} = 1$ is a prior weight.

For all cases, the weight $w_{x'_i}$ is equally set to 1 (Bergstra et al., 2011) or allocated by a recency weighting strategy (Bergstra et al., 2013). MOTPE employs another weighting strategy as described in Section 3.

For more information, we recommend that readers refer to the original papers (Bergstra et al., 2011, 2013) and the implementations of TPE (Bergstra et al., 2015; Akiba et al., 2019) and MOTPE that we provide.[1]

2.3.2 Expected Improvement

To obtain a candidate to evaluate, TPE employs the following Expected Improvement (EI) for y^* and the parameter x_i as the acquisition function (Bergstra et al., 2011):

$$\begin{aligned} \mathrm{EI}_{y^*}(x_i) &:= \int_{-\infty}^{\infty} \max(y^* - y, 0) p(y \mid x_i) dy \\ &= \int_{-\infty}^{y^*} (y^* - y) p(y \mid x_i) dy \\ &\propto \left(\gamma + (1-\gamma)\frac{g(x_i)}{l(x_i)}\right)^{-1}. \end{aligned} \tag{7}$$

In each iteration, for each active parameter, the best candidate x_i^* with the greatest EI value is selected to evaluate (lines 2–9 of Algorithm 1). At first, parameters that connect to the root of the search space are active and sampled, e.g., x_1 and x_2 in Figure 1. After sampling those parameters, some conditional parameters can be active and sampled, e.g., when $x_2^* = 2$, then x_3 and x_4 in Figure 1 become active. In brief, TPE samples the parameters from the root to the leaves of the search space. Usually, the maximizer of EI for each parameter x_i^* is approximated by selecting the best one among several candidates sampled from $l(x_i)$ (lines 7–8). In line 7, when the parameter type of x_i is integer, a sampled candidate is rounded to the nearest integer value. Note that no specific model for $p(y)$ is required since $l(x_i)/g(x_i)$ does not depend on $p(y)$.

1. https://doi.org/10.5281/zenodo.6258358.

3. Multiobjective Tree-Structured Parzen Estimator

In this section, we introduce the MOTPE algorithm to solve the problem (2). This algorithm is an extension of the TPE algorithm for multiobjective optimization.

Assume a set of observations $D = \{(\boldsymbol{x}^{(1)}, \boldsymbol{y}^{(1)}), \ldots, (\boldsymbol{x}^{(k)}, \boldsymbol{y}^{(k)})\}$. MOTPE models $p(x_i \mid \boldsymbol{y})$ for each parameter x_i $(\in X_i)$ using the two probability density functions as follows:

$$p(x_i \mid \boldsymbol{y}) = \begin{cases} l(x_i) & \text{if } (\boldsymbol{y} \prec Y^*) \vee (\boldsymbol{y} \parallel Y^*) \\ g(x_i) & \text{if } Y^* \preceq \boldsymbol{y} \end{cases} \tag{8}$$

where Y^* is a set of objective vectors such that $p((\boldsymbol{y} \prec Y^*) \vee (\boldsymbol{y} \parallel Y^*)) = \gamma$. The $\gamma \in (0, 1)$ is a quantile parameter, $l(x_i)$ is a density constructed using the subset of the observed x_i values $D_{l_x} = \{x_i^{(j)} \in X_i \mid (\boldsymbol{y}^{(j)} (= \boldsymbol{f}(\boldsymbol{x}^{(j)})) \prec Y^*) \vee (\boldsymbol{y}^{(j)} \parallel Y^*),\ j = 1, \ldots, k\}$, and $g(x_i)$ is a density constructed using the set of the remaining observed x_i values D_{g_x}. Figure 2 illustrates the relationship between $\boldsymbol{y}$ and Y^* in a two-dimensional case. MOTPE constructs the model like TPE (cf. Section 2.3.1). A simple but significant difference between the models of MOTPE and TPE is the weighting strategy. MOTPE assigns a weight proportional to the hypervolume contribution for each observation for $l(x_i)$ and a uniform weight for each observation for $g(x_i)$. That is, the weight for $x_i^{(j)} \in D_{l_x}$ is set to be proportional to $I_{\mathrm{H}}(D_{l_{\boldsymbol{y}}}) - I_{\mathrm{H}}(D_{l_{\boldsymbol{y}}} \setminus \{\boldsymbol{y}^{(j)}\})$ where $D_{l_{\boldsymbol{y}}} = \{\boldsymbol{y}^{(j)} \mid x_i^{(j)} \in D_{l_x}\}$ and the weight for $x_i^{(j)} \in D_{g_x}$ is set to 1.

3.1 Greedy Algorithm to Split Observations

MOTPE utilizes a greedy algorithm to split observations including many incomparable ones. The MOTPE algorithm theoretically depends on Y^* such that $p((\boldsymbol{y} \prec Y^*) \vee (\boldsymbol{y} \parallel Y^*)) = \gamma$ to split the observations into subsets for $l(x_i)$ and $g(x_i)$. However, in practice, the observations are directly split for a specific γ in a greedy manner, as described in Algorithm 2. In Algorithm 2, $D_{\mathrm{rank}(j)}$ denotes the set of observations with the objective vectors having the nondomination rank j. This algorithm comprises the following two steps. The first step (lines 3–6) greedily appends better nondomination ranked observations to the largest

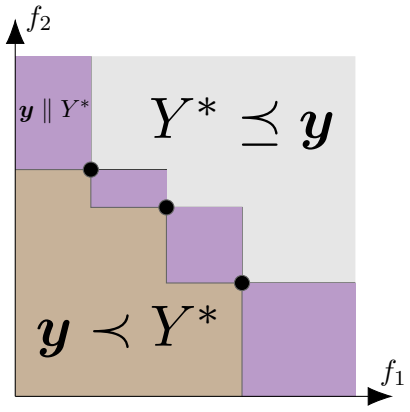


Figure 2: The relationship between $\boldsymbol{y}$ and Y^*. Black points belong to Y^*.

Algorithm 2 Split Observations

Require:
$D = \{(\boldsymbol{x}^{(1)}, \boldsymbol{y}^{(1)}), \ldots, (\boldsymbol{x}^{(k)}, \boldsymbol{y}^{(k)})\}$: observations
$\gamma \in (0, 1)$: quantile

1: $D_l \leftarrow \{\}$
2: $j \leftarrow 1$
3: **while** $|D_l| + |D_{\mathrm{rank}(j)}| \leq \lfloor \gamma |D| \rfloor$ **do**
4: $\quad D_l \leftarrow D_l \cup D_{\mathrm{rank}(j)}$
5: $\quad j \leftarrow j + 1$
6: **end while**
7: $D_l \leftarrow D_l \cup \textsc{greedy\_hss}(D_{\mathrm{rank}(j)}, \lfloor \gamma |D| \rfloor - |D_l|)$ ▷ call Algorithm 3
8: $D_g \leftarrow D \setminus D_l$
9: **return** (D_l, D_g)

Algorithm 3 Greedy Hypervolume Subset Selection

Require:
D_{i} : input set
$n_{\mathrm{s}} \in \mathbb{N}$: subset size

1: $D_{\mathrm{s}} \leftarrow \{\}$
2: **for all** $(\boldsymbol{x}, \boldsymbol{y}) \in D_{\mathrm{i}}$ **do**
3: $\quad c_{\boldsymbol{y}} \leftarrow I_{\mathrm{H}}(\{\boldsymbol{y}\})$
4: **end for**
5: **for** $j \leftarrow 1, \ldots, n_{\mathrm{s}}$ **do**
6: $\quad (\boldsymbol{x}', \boldsymbol{y}') \leftarrow \mathrm{argmax}_{(\boldsymbol{x}, \boldsymbol{y}) \in D_{\mathrm{i}}} c_{\boldsymbol{y}}$
7: $\quad D_{\mathrm{i}} \leftarrow D_{\mathrm{i}} \setminus \{(\boldsymbol{x}', \boldsymbol{y}')\}$
8: $\quad$ **for all** $(\boldsymbol{x}, \boldsymbol{y}) \in D_{\mathrm{i}}$ **do**
9: $\quad\quad c_{\boldsymbol{y}} \leftarrow I_{\mathrm{H}}(D_{\mathrm{s}} \cup \{(\max(y_1, y'_1), \ldots, \max(y_m, y'_m))\}) - I_{\mathrm{H}}(D_{\mathrm{s}})$
10: $\quad$ **end for**
11: $\quad D_{\mathrm{s}} \leftarrow D_{\mathrm{s}} \cup \{(\boldsymbol{x}', \boldsymbol{y}')\}$
12: **end for**
13: **return** D_{s}

extent possible to D_l. This only requires the nondominated sorting (Deb et al., 2002). The second step (line 7) appends the set obtained as a result of the hypervolume subset selection problem (HSSP) (Bader & Zitzler, 2011) to D_l. HSSP finds a subset of a specific size of a set of vectors to maximize the hypervolume indicator with a given reference point. Because HSSP is a submodular maximization, it can obtain a near-optimal solution having the hypervolume that is lower bounded by $1 - 1/e$ (e is Napier's constant) times the optimal value via the greedy method (Nemhauser, Wolsey, & Fisher, 1978; Guerreiro, Fonseca, & Paquete, 2016). Here we describe a general greedy algorithm (Bradstreet, While, & Barone, 2007; Guerreiro et al., 2016) for HSSP in Algorithm 3. This algorithm requires a set D_{i} and size n_{s} and returns the result of HSSP for these inputs. In our experiments, the vector $(1.1 \max_{\boldsymbol{y} \in D_{\boldsymbol{y}}} y_1, \ldots, 1.1 \max_{\boldsymbol{y} \in D_{\boldsymbol{y}}} y_m)$ where $D_{\boldsymbol{y}} = \{\boldsymbol{y} \mid (\boldsymbol{x}, \boldsymbol{y}) \in D_{\mathrm{rank}(j)}\}$, is used as the reference point for the hypervolume indicator (cf. Definition 7) in Algorithm 3.

3.2 Expected Hypervolume Improvement

Because EI cannot be applied to multiobjective optimization, we use instead the Expected Hypervolume Improvement (EHVI) acquisition function (Emmerich, Giannakoglou, & Naujoks, 2006; Emmerich, Deutz, & Klinkenberg, 2011) that measures the expectation of hypervolume improvement. The hypervolume improvement is important because the hypervolume indicator has the following crucial properties (Zitzler, Thiele, Laumanns, Fonseca, & Da Fonseca, 2003; Fleischer, 2003). Let Y and Y' be subsets of $\mathbb{R}^m$. The hypervolume indicator never contradicts Pareto optimality: $I_{\mathrm{H}}(Y) > I_{\mathrm{H}}(Y') \Rightarrow \neg((\forall \boldsymbol{y} \in Y \exists \boldsymbol{y}' \in Y' : \boldsymbol{y}' \preceq \boldsymbol{y}) \wedge (Y \neq Y'))$. The hypervolume indicator can always detect if one set is better than another in terms of Pareto optimality: $(\forall \boldsymbol{y}' \in Y' \exists \boldsymbol{y} \in Y : \boldsymbol{y} \preceq \boldsymbol{y}') \wedge (Y \neq Y') \Rightarrow I_{\mathrm{H}}(Y) > I_{\mathrm{H}}(Y')$. The Pareto front achieves the maximum hypervolume. By utilizing EHVI, MOTPE tries to greedily maximize the hypervolume.

EHVI for Y^* and the parameter x_i is calculated as follows:

$$\begin{aligned}
\mathrm{EHVI}_{Y^*}(x_i) &\coloneqq \int (I_{\mathrm{H}}(Y^* \cup \{\boldsymbol{y}\}) - I_{\mathrm{H}}(Y^*)) p(\boldsymbol{y} \mid x_i) d\boldsymbol{y} \\
&= \int_R (I_{\mathrm{H}}(Y^* \cup \{\boldsymbol{y}\}) - I_{\mathrm{H}}(Y^*)) p(\boldsymbol{y} \mid x_i) d\boldsymbol{y} \\
&= \int_R (I_{\mathrm{H}}(Y^* \cup \{\boldsymbol{y}\}) - I_{\mathrm{H}}(Y^*)) \frac{p(x_i \mid \boldsymbol{y}) p(\boldsymbol{y})}{p(x_i)} d\boldsymbol{y}
\end{aligned} \tag{9}$$

where $R = \{\boldsymbol{y} \mid (\boldsymbol{y} \prec Y^*) \vee (\boldsymbol{y} \parallel Y^*)\}$. By definition in Equation (8), we obtain $p(x_i \mid \boldsymbol{y}) = l(x_i)$ as long as $\boldsymbol{y} \in R$, otherwise $p(x_i \mid \boldsymbol{y}) = g(x_i)$. Based on the definition, $\gamma = p(\boldsymbol{y} \in R)$, and $p(x_i) = \int p(x_i \mid \boldsymbol{y}) p(\boldsymbol{y}) d\boldsymbol{y} = \gamma l(x_i) + (1-\gamma) g(x_i)$; therefore, the numerator of $\mathrm{EHVI}_{Y^*}(x_i)$ is

$$\begin{aligned}
&\int_R (I_{\mathrm{H}}(Y^* \cup \{\boldsymbol{y}\}) - I_{\mathrm{H}}(Y^*)) p(x_i \mid \boldsymbol{y}) p(\boldsymbol{y}) d\boldsymbol{y} \\
&= l(x_i) \underbrace{\int_R (I_{\mathrm{H}}(Y^* \cup \{\boldsymbol{y}\}) - I_{\mathrm{H}}(Y^*)) p(\boldsymbol{y}) d\boldsymbol{y}}_{= C_{Y^*} \text{ (constant w.r.t. } x_i)} \\
&= C_{Y^*} l(x_i),
\end{aligned} \tag{10}$$

and finally, we obtain the following result:

$$\begin{aligned}
\mathrm{EHVI}_{Y^*}(x_i) &= \frac{C_{Y^*} l(x_i)}{p(x_i)} \\
&= \frac{C_{Y^*} l(x_i)}{\gamma l(x_i) + (1-\gamma) g(x_i)} \\
&= \frac{C_{Y^*}}{\gamma + (1-\gamma) \frac{g(x_i)}{l(x_i)}} \\
&\propto \left(\gamma + (1-\gamma) \frac{g(x_i)}{l(x_i)} \right)^{-1}.
\end{aligned} \tag{11}$$

The result means that we can maximize EHVI by simply maximizing the ratio $l(x_i)/g(x_i)$ without calculation of the actual EHVI value. We find that no specific model for $p(\boldsymbol{y})$ is required since $l(x_i)/g(x_i)$ does not depend on $p(\boldsymbol{y})$.

Algorithm 4 Multiobjective Tree-structured Parzen Estimator

Require:
$D = \{(\boldsymbol{x}^{(1)}, \boldsymbol{y}^{(1)}), \ldots, (\boldsymbol{x}^{(k)}, \boldsymbol{y}^{(k)})\}$: observations
$n_{\mathrm{t}} \in \mathbb{N}$: number of iterations
$n_{\mathrm{c}} \in \mathbb{N}$: number of candidates
$\gamma \in (0, 1)$: quantile
1: **for** $t \leftarrow 1, \ldots, n_{\mathrm{t}}$ **do**
2: $(D_l, D_g) \leftarrow$ SPLIT_OBSERVATIONS(D, γ) ▷ call Algorithm 2
3: **repeat**
4: $i \leftarrow i \in [1, n]$ such that x_i is active and x_i^* has not been sampled
5: construct $l(x_i)$ with x_i values in D_l and $g(x_i)$ with x_i values in D_g
6: $C_i \leftarrow \{x_i^{(j)} \sim l(x_i) \mid j = 1, \ldots, n_{\mathrm{c}}\}$ ▷ sample n_{c} candidates for x_i^*
7: $x_i^* \leftarrow \mathrm{argmax}_{x_i \in C_i} l(x_i)/g(x_i)$ ▷ approximate $\mathrm{argmax}_{x_i \in X_i} l(x_i)/g(x_i)$
8: **until** all active parameters have been sampled
9: $D \leftarrow D \cup \{(\boldsymbol{x}^*, \boldsymbol{f}(\boldsymbol{x}^*))\}$ ▷ $\boldsymbol{x}^*$ is the vector composed of all sampled x_i^*
10: **end for**
11: **return** the set of nondominated solutions in D

The pseudocode of MOTPE is provided in Algorithm 4. In each iteration, for each parameter, the best candidate $x_i^* = \mathrm{argmax}_{x_i \in X_i} \mathrm{EHVI}_{Y^*}(x_i)$ will be selected to evaluate.[2] As same as TPE, this maximizer of EHVI is approximated by a limited number of samples (see lines 6–7). In line 6, when the parameter type of x_i is integer, a sampled candidate is rounded to the nearest integer value.

3.3 Parallelization

When objectives are computationally expensive, parallelization of evaluating those objectives is a matter of great significance in practice. An asynchronous parallelization method, proposed by Bergstra et al. (2011) for the original TPE, can also be applied to MOTPE without any modification. Of course, it is also able to parallelize the initialization step of MOTPE asynchronously. Algorithm 5 describes the asynchronous parallel MOTPE including initialization. This approach simply ignores unfinished evaluations and draws from $l(x_i)$ to provide different candidates relying on the stochasticity of draws. In line 8 of Algorithm 5, a process (or thread) to run Algorithm 6 is created. In line 2 of Algorithm 6, an arbitrary sampling method such as uniform random sampling or Latin hypercube sampling (McKay, Beckman, & Conover, 1979) can be used as INITIAL_SAMPLING_METHOD.

The asynchronous parallelization is speedy and more efficient without wait time for synchronization (Kandasamy, Krishnamurthy, Schneider, & Póczos, 2018) although synchronous parallelization is widely adopted in evolutionary computing methods and batch Bayesian optimization methods. Figure 3 illustrates the difference between synchronous and asynchronous parallelization. Additionally, in synchronous parallelization of Bayesian optimization, a set of observations to construct a surrogate is always the same among work-

2. In MOTPE, the solution obtained by maximizing EHVI is equivalent to that obtained by maximizing the probability of hypervolume improvement (PHVI). Details of this derivation can be found in Appendix A.

Algorithm 5 Asynchronous Parallel MOTPE

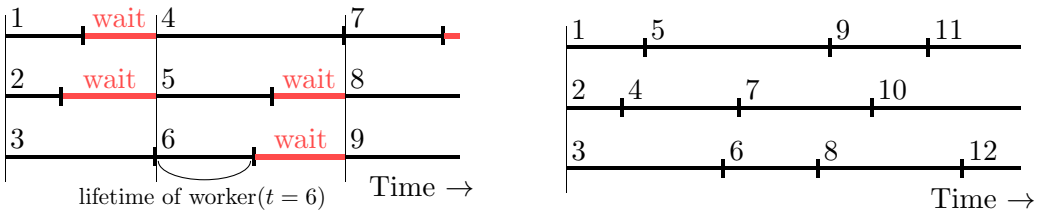
Require:
$n_{\mathrm{t}} \in \mathbb{N}$: number of evaluations (including initialization, i.e., $n_{\mathrm{t}} \geq n_{\mathrm{i}}$)
$n_{\mathrm{i}} \in \mathbb{N}$: number of initial observations
$n_{\mathrm{c}} \in \mathbb{N}$: number of candidates
$n_{\mathrm{p}} \in \mathbb{N}$: number of maximum workers
$\gamma \in (0, 1)$: quantile

$D \leftarrow \{\}$ ▷ observations (shared variable)
$n_{\mathrm{w}} \leftarrow 0$ ▷ number of workers running (shared variable)
for $t \leftarrow 1, \ldots, n_{\mathrm{t}}$ **do**
  if $n_{\mathrm{w}} = n_{\mathrm{p}}$ **then**
    wait until $n_{\mathrm{w}} < n_{\mathrm{p}}$
  end if
  $n_{\mathrm{w}} \leftarrow n_{\mathrm{w}} + 1$
  launch WORKER(t, D, n_{w}, n_{i}, n_{c}, γ) ▷ run Algorithm 6 as a new process (or thread)
end for
wait until $n_{\mathrm{w}} = 0$
return the set of nondominated solutions in D

Algorithm 6 Worker for Asynchronous Parallel MOTPE

Require:
$t \in \mathbb{N}$: evaluation id
D : observations defined in Algorithm 5 (shared variable)
$n_{\mathrm{w}} \in \mathbb{N}$: number of workers running defined in Algorithm 5 (shared variable)
$n_{\mathrm{i}} \in \mathbb{N}$: number of initial observations
$n_{\mathrm{c}} \in \mathbb{N}$: number of candidates
$\gamma \in (0, 1)$: quantile

if $t \leq n_{\mathrm{i}}$ **then**
  $\boldsymbol{x}^* \leftarrow$ INITIAL_SAMPLING_METHOD() ▷ call an arbitrary sampling method
else
  $(D_l, D_g) \leftarrow$ SPLIT_OBSERVATIONS(D, γ) ▷ call Algorithm 2
  repeat
    $i \leftarrow i \in [1, n]$ such that x_i is active and x_i^* has not been sampled
    construct $l(x_i)$ with x_i values in D_l and $g(x_i)$ with x_i values in D_g
    $C_i \leftarrow \{x_i^{(j)} \sim l(x_i) \mid j = 1, \ldots, n_{\mathrm{c}}\}$ ▷ sample n_{c} candidates for x_i^*
    $x_i^* \leftarrow \mathrm{argmax}_{x_i \in C_i}\, l(x_i)/g(x_i)$ ▷ approximate $\mathrm{argmax}_{x_i \in X_i}\, l(x_i)/g(x_i)$
  until all active parameters have been sampled
end if
$D \leftarrow D \cup \{(\boldsymbol{x}^*, \boldsymbol{f}(\boldsymbol{x}^*))\}$ ▷ $\boldsymbol{x}^*$ is the vector composed of all sampled x_i^*
$n_{\mathrm{w}} \leftarrow n_{\mathrm{w}} - 1$



(a) Synchronous parallelization.

(b) Asynchronous parallelization.

Figure 3: Synchronous parallelization vs. Asynchronous parallelization. The number of maximum workers $n_{\mathrm{p}} = 3$. Each number corresponds to the evaluation id t in Algorithm 6 and indicates the launch of worker(t). Each black interval indicates the lifetime of worker(t). Red intervals indicate wait time. In synchronous parallelization, it is needed to wait for finishing all workers currently running to launch a new worker, e.g., worker($t = 4, 5$) cannot be launched before worker($t = 3$) finished running even if worker($t = 1, 2$) finished running. In asynchronous parallelization, a new worker can be launched whenever the number of workers currently running becomes less than n_{p} so that no wait time occurs.

ers per batch. Because of this, batch Bayesian optimization methods require a nontrivial candidate selection approach to make the diversity of candidates (Azimi, Fern, & Fern, 2010; González, Dai, Hennig, & Lawrence, 2016). On the other hand, in asynchronous parallelization of Bayesian optimization, a set of observations to construct a surrogate is rarely the same among workers so that any special effort to make the diversity of candidates is not needed. For these reasons, asynchronous parallelization is generally more suitable for expensive optimization as long as each observation can be obtained one by one. We evaluate the effectiveness of parallelization in Section 5.2.2.

4. Related Work

In the machine learning community, algorithms for expensive multiobjective optimization have been studied as multiobjective Bayesian optimization. PESMO (Hernández-Lobato et al., 2016) and MESMO (Belakaria et al., 2019) utilize an information-theoretic acquisition function. PESMO, which is a multiobjective extension of Predictive Entropy Search (Hernández-Lobato, Hoffman, & Ghahramani, 2014), iteratively evaluates a candidate that maximizes the information gained about the Pareto set. Similarly, MESMO, which is a multiobjective extension of Max-value Entropy Search (Wang & Jegelka, 2017)), iteratively evaluates a candidate that maximizes the information gained about the Pareto front. Both of these methods were applied to HPO and effectively found models with high accuracy and low prediction time compared to baseline methods such as ParEGO (Knowles, 2006) and SMS-EGO (Ponweiser, Wagner, Biermann, & Vincze, 2008). Especially, PESMO is available in Spearmint (Snoek, Larochelle, & Adams, 2012), which is a famous open-source software for HPO. Active learning approaches for multiobjective optimization are also proposed (Zuluaga, Sergent, Krause, & Püschel, 2013; Campigotto, Passerini, & Battiti, 2013; Zuluaga, Krause, & Püschel, 2016). One of the active learning approaches is ALP (Campig-

otto et al., 2013). ALP generates the analytical Pareto front representation by learning a model from an approximated Pareto set. There are also several recent studies about multiobjective Bayesian optimization (Picheny, 2015; Shah & Ghahramani, 2016; Feliot, Bect, & Vazquez, 2017; Abdolshah, Shilton, Rana, Gupta, & Venkatesh, 2019; Parsa, Mitchell, Schuman, Patton, Potok, & Roy, 2020). We remark that all of the above studies use GP or its extension as a surrogate. Besides popular GP, there is a limited number of Bayesian optimization algorithms with a non-GP surrogate. Recently, Nardi, Souza, Koeplinger, and Olukotun (2019) proposed HyperMapper 2.0 for the automatic static tuning of hardware accelerators. HyperMapper 2.0 employs random forests so that it can handle continuous, discrete, and categorical variables. However, the algorithm has not been tested on problems with a tree-structured search space.

In the evolutionary computation community, many surrogate-assisted multiobjective evolutionary algorithms have been proposed (Chugh et al., 2019). Knowles (2006) proposed ParEGO, which is an extension of the single-objective efficient global optimization (EGO) algorithm (Jones, Schonlau, & Welch, 1998). ParEGO reduces a multiobjective optimization problem into a single-objective optimization problem via an augmented Tchebycheff aggregation. It utilizes the design and analysis of computer experiments (DACE) model (Sacks, Welch, Mitchell, & Wynn, 1989) based on GP to approximate the objective landscape and evaluates the candidate solution with the maximum EI value. Ponweiser et al. (2008) proposed SMS-EGO, which is another extension of the EGO algorithm. SMS-EGO also utilizes the DACE model; however, it differs from ParEGO in that the candidate is determined based on the hypervolume contribution in each iteration. Emmerich et al. (2006, 2011) proposed an approach to calculate EI in hypervolume (i.e., EHVI), which is a natural extension of single-objective EI acquisition function for multiobjective optimization. Besides EHVI, various acquisition functions have been proposed for multiobjective optimization, e.g., EIPBI, EIPBII, MPoI, HypI, DomRank, MSD, and EIM (Namura, Shimoyama, & Obayashi, 2017; Rahat, Everson, & Fieldsend, 2017; Zhan, Cheng, & Liu, 2017). Zhang, Liu, Tsang, and Virginas (2009) proposed MOEA/D-EGO combining MOEA/D (Zhang & Li, 2007) with GP. Chugh et al. (2018), Chugh, Sun, Wang, and Jin (2020) proposed K-RVEA combining RVEA (Cheng, Jin, Olhofer, & Sendhoff, 2016) with GP. K-RVEA has been successful in the air intake ventilation system design (Chugh et al., 2017). In surrogate-assisted evolutionary algorithms, non-GP surrogates are also sometimes employed (Chugh et al., 2019). Recent examples of non-GP-based algorithms are CSEA (Pan, He, Tian, Wang, Zhang, & Jin, 2018) and HeE-MOEA (Guo, Jin, Ding, & Chai, 2018). CSEA employs a feedforward neural network classifier to predict the dominance relationship between candidate solutions and reference solutions. HeE-MOEA employs an ensemble of radial basis function networks and least square support vector machines. These studies pointed out the scalability issue of GP and they proposed more scalable surrogates to the number of training samples and the dimension of the search space. However, these studies do not address solving problems with non-continuous and/or conditional variables.

Finally, we refer to an early interesting work MOPED (Costa & Minisci, 2003), which is a multiobjective evolutionary algorithm based on the Parzen estimator. MOTPE and MOPED similarly use a Parzen estimator-based model. However, there are important differences between MOTPE and MOPED. First, MOTPE exploits the information of both good and poor observations through the EHVI acquisition function to efficiently sample

promising solutions with a limited evaluation budget whereas MOPED only exploits the information of good observations. This leads to MOTPE having better sample efficiency than MOPED. Second, MOTPE is designed to solve problems with a tree-structured search space including non-continuous variables whereas MOPED is for continuous optimization. Therefore, MOTPE can be applied to a wider range of problems than MOPED. We consider that these differences are enough reasons to use MOTPE rather than MOPED for problems that we address.

To our knowledge, any non-GP-based algorithm has not been in a dominant position as a suitable optimizer for expensive multiobjective problems in a complex search space like TPE in single-objective optimization.

5. Numerical Results

In this section, we describe a set of experiments and discuss the results. The results show that MOTPE outperforms other methods and has a sufficient ability for practical use. We also investigate the behavior of MOTPE with different γ settings and parallelization.

5.1 Comparisons with Other Methods

First, we test MOTPE on a number of well-designed benchmark problems with a limited evaluation budget to analyze the fundamental performance of MOTPE. MOTPE is compared with Bayesian optimization methods: ParEGO (Knowles, 2006), SMS-EGO (Ponweiser et al., 2008), and PESMO (Hernández-Lobato et al., 2016). Then, we compare MOTPE and NSGA-II (Deb et al., 2002) on the same benchmark problems with medium dimension and medium budget settings to demonstrate the scalability of MOTPE. Finally, we compare MOTPE, ParEGO, SMS-EGO, PESMO, and HyperMapper 2.0 (Nardi et al., 2019) on a multiobjective CNN design problem, which is an important real-world application. We implemented MOTPE[3] by modifying the TPE implementation of Optuna (version 2.0.0) (Akiba et al., 2019) whereas we used Spearmint[4] (Snoek et al., 2012) for ParEGO, SMS-EGO and PESMO, and HyperMapper 2.0 (version 2.2.3) for Bayesian optimization with a random forests-based surrogate because Optuna does not provide these algorithms.

5.1.1 WFG Benchmark: Low Dimension & Limited Budget Setting

The WFG benchmark suite (Huband, Barone, While, & Hingston, 2005; Huband, Hingston, Barone, & While, 2006) that consists of nine problems was used to analyze the fundamental performance of MOTPE. The WFG benchmark problems define four parameters as the number of objectives m, the number of variables n, the number of position-related parameters k, and the number of distance-related parameters l. Regarding the working parameters k and l, k must be a multiple of $m-1$, and $k+l$ must be equal to n (Huband et al., 2006). Additionally, l must be a multiple of 2 for WFG2 and WFG3. We prepared the following three settings for the nine benchmark problems.

3. The code is available at https://doi.org/10.5281/zenodo.6258358. The singularity container image which we used to run our code and the experimental data of Section 5 are available upon request.
4. Spearmint's PESM branch: https://github.com/HIPS/Spearmint/tree/PESM. We used the latest version updated on July 26, 2016.

Parameter	Value
likelihood	NOISELESS
grid_size	1,000
moo_grid_size_to_solve_problem	1,000
moo_use_grid_only_to_solve_problem	true
pesm_use_grid_only_to_solve_problem	true
pesm_pareto_set_size	50
pesm_not_constrain_predictions	false

Table 1: Spearmint settings.

(1) Toy: $(m = 2, n = 3, k = 1, l = 2)$

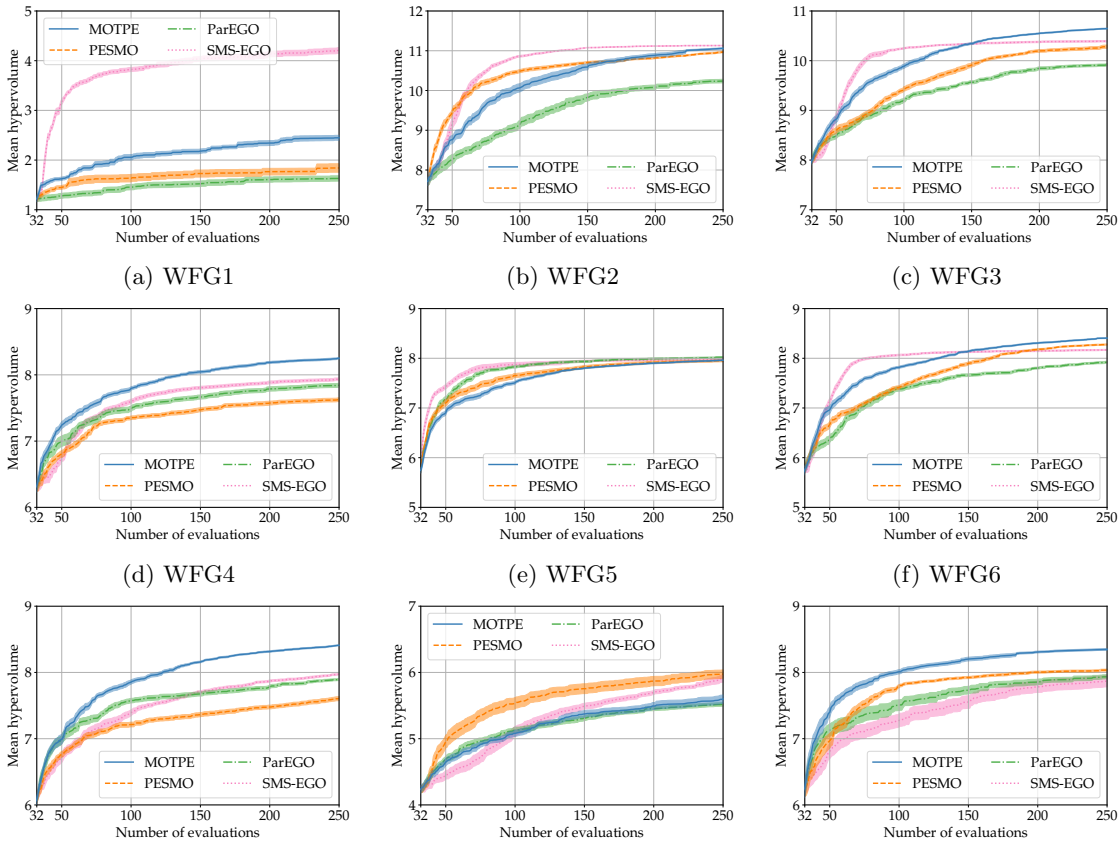
(2) Practical: $(m = 2, n = 9, k = 1, l = 8)$

(3) Many-objective: $(m = 4, n = 9, k = 3, l = 6)$.

All WFG problems have a search space of each variable x_i in the range $[0, 2i]$. The evaluation budget was set to 250 for all compared algorithms, including initial evaluations. This setting simulates expensive objectives that cannot be evaluated many times in practice. The number of initial observations n_{i} was set to $11n-1$ (i.e., 32 for $n = 3$ and 98 for $n = 9$), and the initial solutions were sampled using the Latin hypercube sampling (McKay et al., 1979). This initialization procedure has been commonly used in existing studies (Knowles, 2006; Ponweiser et al., 2008; Chugh et al., 2018). We set $\gamma = 0.10$, $n_{\mathrm{c}} = 24$, and the scales for all parameters to uniform for MOTPE. The setting $n_{\mathrm{c}} = 24$ follows the default setting of TPE in HyperOpt (Bergstra et al., 2011) and Optuna (Akiba et al., 2019). The settings for Spearmint are shown in Table 1. These settings are mostly based on the online article by the author of PESMO.[5] We have changed the likelihood parameter to NOISELESS since the WFG benchmarks are noiseless. For each method and setting, 21 optimization runs were performed to get a mean hypervolume and a standard error. As a metric of performance, the hypervolume indicator with the reference point $(3, 5)$ (resp. $(3, 5, 7, 9)$) for $m = 2$ (resp. $m = 4$) was used. We used this reference point because $\max_{\boldsymbol{x} \in X} f_j(\boldsymbol{x}) = 2j + 1$ $(j = 1, \ldots, m)$ holds for all WFG problems (Huband et al., 2006), i.e., it is inherently equipped with the theoretically worst value for each objective.

Figures 4–7 show the obtained results. Note that the results of initialization that are the same for all methods are omitted in Figures 4, 5, and 7. As can be seen in these figures, MOTPE achieves results comparable to or better than those of the other methods on the WFG problems except for WFG1 and WFG5. These results indicate that MOTPE is relatively robust to separability, modality, and Pareto front geometry of problems (Huband et al., 2006). Comparing the results of Figure 4 and Figure 5, we find that the performance of MOTPE did not so deteriorate while that of the other methods deteriorated. This indicates that modeling based on the parameter-wise Parzen estimators is more robust to the dimensionality than GP that directly approximates a high-dimensional objective landscape.

5. Neural networks with optimal accuracy and speed in their predictions: https://towardsdatascience.com/neural-networks-with-optimal-accuracy-and-speed-in-their-predictions-d2cdc3b21b50.



(a) WFG1 (b) WFG2 (c) WFG3

(d) WFG4 (e) WFG5 (f) WFG6

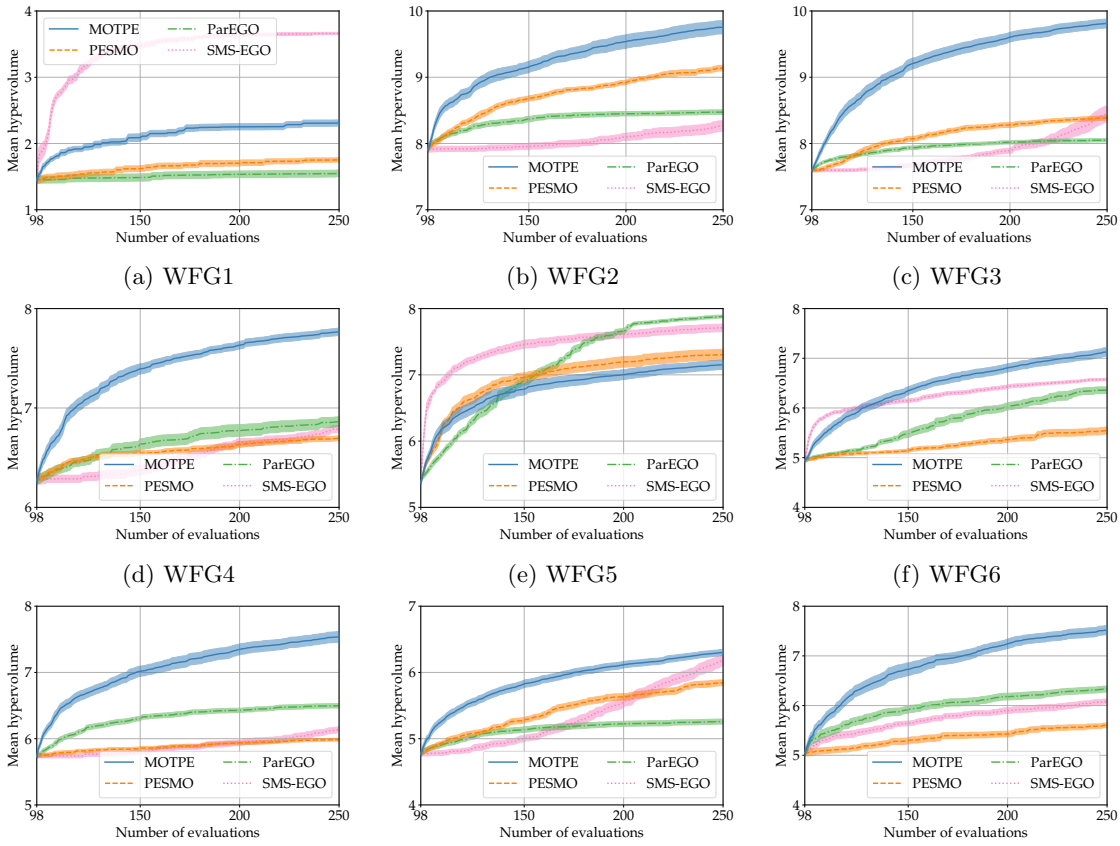
(g) WFG7 (h) WFG8 (i) WFG9

Figure 4: Mean hypervolume ± standard error: $(m = 2, n = 3, k = 1, l = 2)$.

Comparing the results of Figure 5 and Figure 7, the trends are similar. This shows that MOTPE scales to at least four objectives without serious degradation. Each optimization run, i.e., a set of 250 evaluations for a single task, took several minutes for MOTPE, several hours for ParEGO and PESMO, and 1–2 days for SMS-EGO. Potential reasons for this considerable difference are the following. First, Parzen estimators are computationally inexpensive compared with GP. Second, MOTPE maximizes the acquisition function in a more simplified way, which accordingly leads to a run time reduction. Contrarily, PESMO requires a more complex approximation procedure (Hernández-Lobato et al., 2016) and SMS-EGO requires expensive hypervolume calculation (Ponweiser et al., 2008).

In summary, we conclude that the performance of MOTPE is promising. MOTPE outperforms the GP-based methods when a search space is continuous and relatively low dimensional, i.e., extremely tractable for GP. The strategy of MOTPE works well in many problems except for the case that a target problem is extremely biased or highly deceptive. And, the GP-based methods also cannot solve an extremely biased problem well and are deceived by a highly deceptive problem. We provide a detailed discussion in the following.

WFG1 is one of the two problems that MOTPE achieved poor results, and in fact, it is a difficult one as well for all the other three methods. This problem is separable and



(a) WFG1 (b) WFG2 (c) WFG3

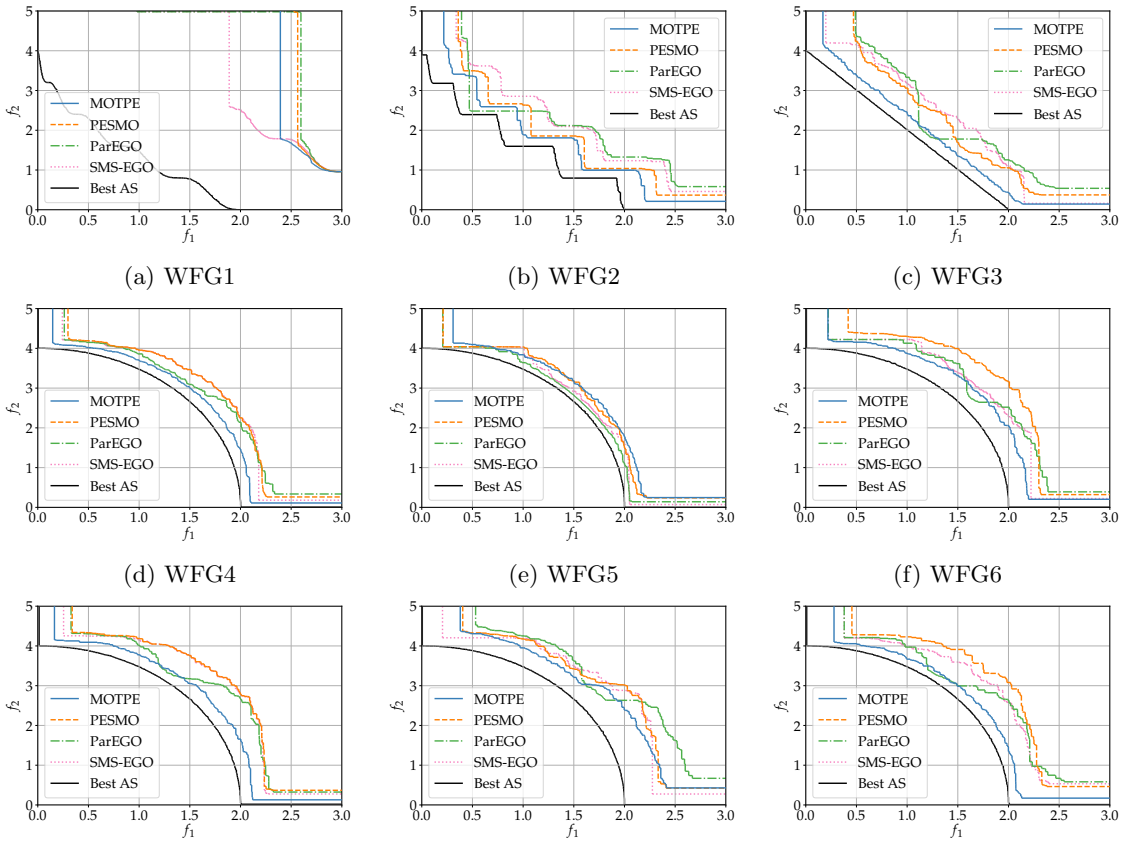
(d) WFG4 (e) WFG5 (f) WFG6

(g) WFG7 (h) WFG8 (i) WFG9

Figure 5: Mean hypervolume ± standard error: $(m = 2, n = 9, k = 1, l = 8)$.

unimodal (Huband et al., 2006). Here, a separable problem means that its objectives can be optimized by considering each parameter in turn, independently of one another. However, this problem is also extremely biased, i.e., evenly distributed points in the search space are not mapped to evenly distributed points in the objective space (Huband et al., 2006). Indeed, we find that most observations are in the lower-right area of Figure 6a despite the initialization with the Latin hypercube sampling. In such a situation, the strategy of MOTPE becomes ineffective at estimating densities of good and poor observations. The results of WFG1 lead to the following lessons. If there is not enough diversity in the objective space at the end of initialization, MOTPE cannot be suitable for the problem. Figure 6a also shows that PESMO and ParEGO are worse than MOTPE under the extremely biased condition. Among the four methods, SMS-EGO is empirically the least susceptible to bias.

WFG2 has a disconnected true Pareto front that consists of 5 separate regions. It is nonseparable and multimodal except its first objective, but not biased so that diversity in the objective space can be easily obtained. Figure 6b shows that MOTPE converges faster than the other methods while maintaining sufficient diversity. Therefore, unlike bias, nonseparability and multimodality of the problem seem to be not so problematic for MOTPE.



(a) WFG1 (b) WFG2 (c) WFG3

(d) WFG4 (e) WFG5 (f) WFG6

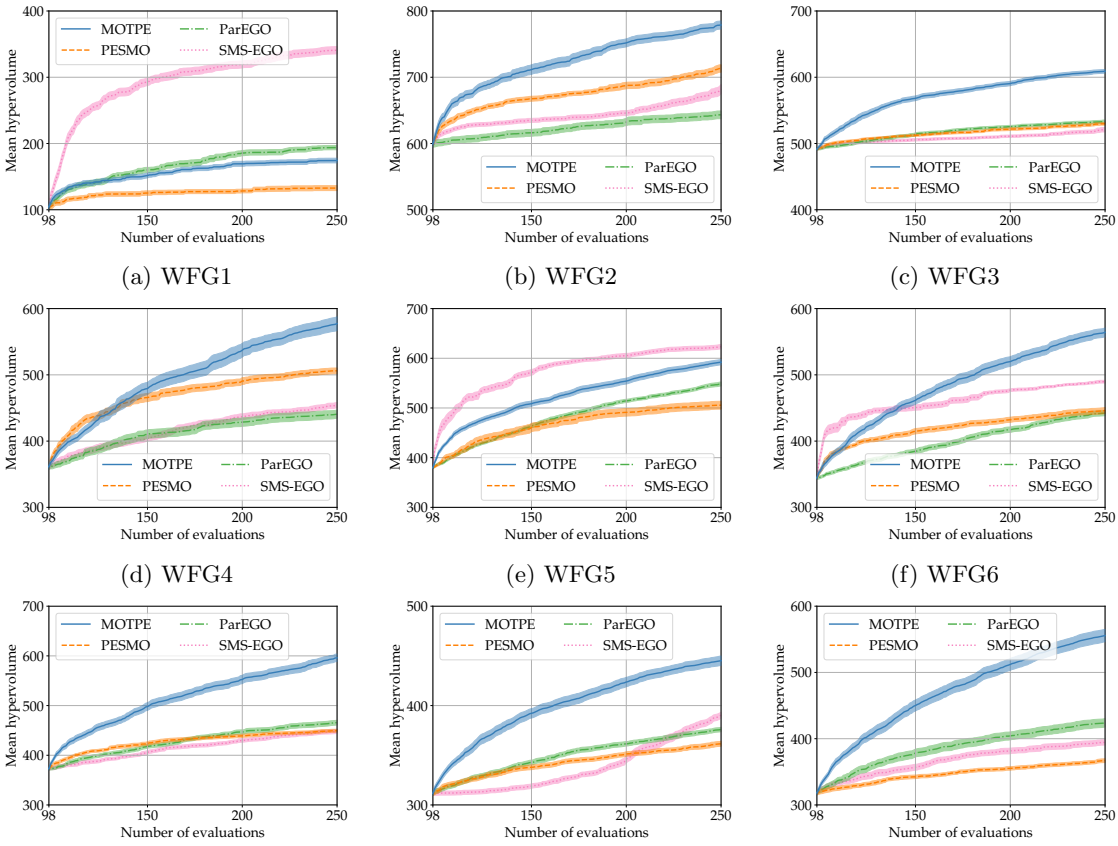
(g) WFG7 (h) WFG8 (i) WFG9

Figure 6: 50%-attainment surface: $(m = 2, n = 9, k = 1, l = 8)$. A 50%-attainment surface denotes the boundary of the points that have been weakly dominated at least 50% of the runs (López-Ibánez et al., 2010). For each problem, the boundary of the objective space $\mathbb{R}^m$ that has been weakly dominated by the Pareto Front corresponding to the Pareto set approximated by $100,000$ Pareto optimal solutions is shown as a black line (Best AS).

WFG3 is an unbiased, nonseparable, and unimodal problem with a linear degenerated true Pareto front. The results show that this problem is not so difficult for MOTPE. We find that the convergence speed of the method is quite fast in Figure 6c.

WFG4 is an unbiased, separable, and multimodal problem with a concave true Pareto front. The results show that concavity is not problematic for MOTPE. Similar to the case of WFG3, the faster convergence of MOTPE is found in Figure 6d.

WFG5 is one of the two problems that MOTPE achieved poor results. This problem is unbiased, separable, and unimodal but highly deceptive (Huband et al., 2006). In general, it is difficult for MOTPE to find a true optimum for a highly deceptive objective for the following reasons. First, since MOTPE samples a new candidate to evaluate from $l(x_i)$, a candidate near a true optimum is rarely sampled unless there is at least one observation



(a) WFG1 (b) WFG2 (c) WFG3

(d) WFG4 (e) WFG5 (f) WFG6

(g) WFG7 (h) WFG8 (i) WFG9

Figure 7: Mean hypervolume ± standard error: $(m = 4, n = 9, k = 3, l = 6)$.

classified into good ones relatively near a true optimum. Second, even if there is such observation, if there are many poor observations around a true optimum, a candidate near a true optimum will have a small EHVI value and the candidate will not be selected. However, this is not the reason why MOTPE was worse than the other methods on WFG5. In fact, the other three methods also failed to find true optima. One of the actual reasons seems to lie in the ability to find good solutions of WFG5 existing at the boundary of the search space. To sample a real-valued parameter, MOTPE draws candidate values from a truncated Gaussian mixture. This sampling procedure based on the original TPE implementation is practical in most cases but difficult to sample candidates on boundaries. In Appendix B, we demonstrate that modifying MOTPE to make it easier to sample boundary points actually improves the results on WFG5 to some degree but also worsens the results on some other problems.

WFG6 is an unbiased, nonseparable, and unimodal problem with a concave true Pareto front. These characteristics do not deteriorate the performance of MOTPE so much. The results show that MOTPE achieved the best hypervolume of all the methods.

WFG7 is a separable and unimodal problem with a concave true Pareto front. In this problem, the fitness landscapes of the position-related parameters are dependent on other

parameters including distance-related parameters (Huband et al., 2006). These dependencies make it difficult to obtain a widely spread Pareto optimal solution set. Based on the results, the performance of MOTPE did not deteriorate by these characteristics.

WFG8 is a nonseparable and unimodal problem with a concave true Pareto front. In this problem, the fitness landscapes of the distance-related parameters are dependent on other parameters including position-related parameters (Huband et al., 2006). As a consequence of the dependencies, Pareto optimal solutions have different distance-related parameter values. Our results show that these characteristics relatively deteriorate the performance of MOTPE.

WFG9 is a nonseparable, multimodal, and deceptive problem with a concave true Pareto front. In this problem, the position-related parameters and the distance-related parameters are interdependent. For this problem, MOTPE achieved results similar to WFG5. On the other hand, the other three methods achieved worse results than those of WFG5. As a result, MOTPE achieved the best performance of all.

5.1.2 WFG Benchmark: Medium Dimension & Medium Budget Setting

In the previous experiment, we have confirmed the performance of MOTPE in low-dimensional problems with a limited budget. Then, we briefly show the scalability of MOTPE to the number of variables and evaluations. In the following experiment, NSGA-II (Deb et al., 2002) was used as a baseline instead of the GP-based methods that we have used so far because of the following reasons: (1) NSGA-II is scalable to the number of variables and evaluations whereas the GP-based methods are not so, i.e., the latter cannot finish running within reasonable time for medium-dimensional problems with a medium evaluation budget, (2) NSGA-II is one of the de facto standard baselines in the multi-objective optimization community because of its popularity, (3) NSGA-II shares the important property with MOTPE, that is, these algorithms utilize the dominance relationship, and (4) when comparing the elapsed times of methods, it is preferable to implement all the methods to be compared in the same software framework. For the reason (4), the NSGA-II implementation of Optuna (version 2.0.0) was used.

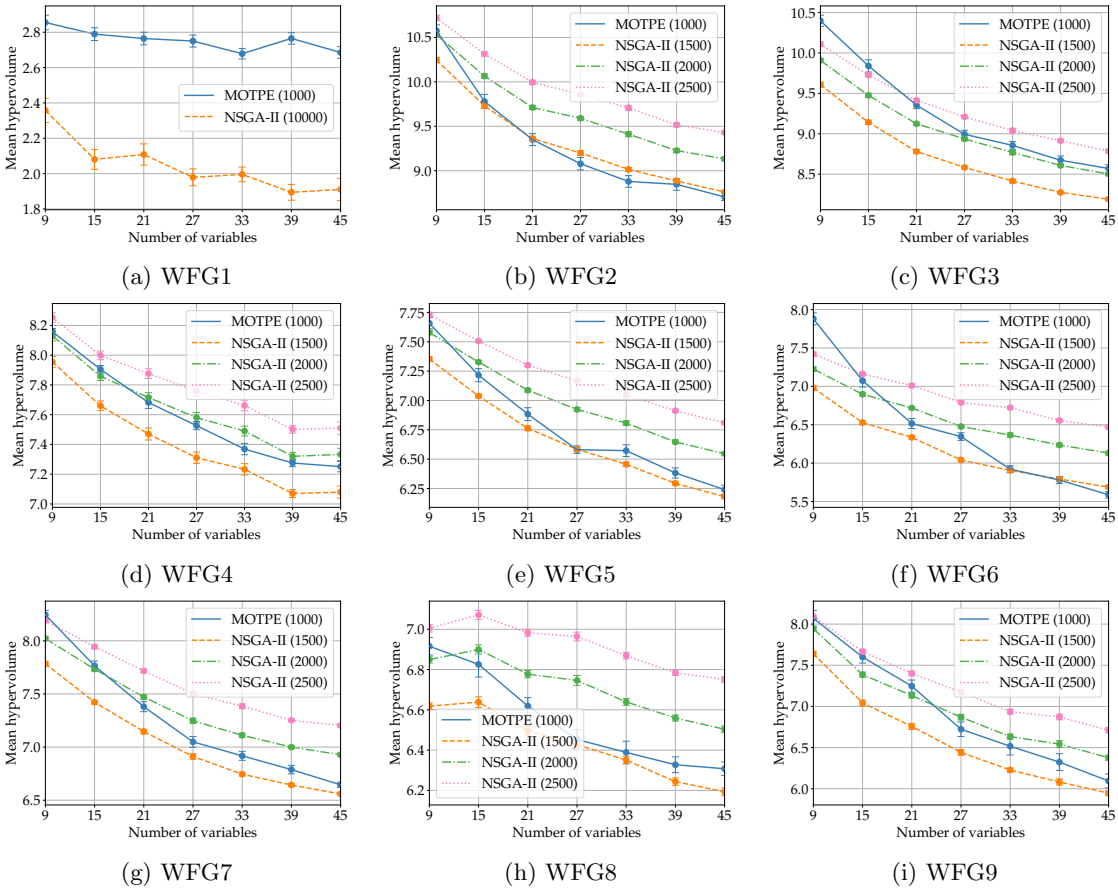
We used the WFG benchmark suite again and prepared the following settings for $n = 9, 15, 21, 27, 33, 39, 45$.

(4) Medium-dimensional: $(m = 2, n, k = 1, l = n - 1)$.

(5) Medium-dimensional many-objective: $(m = 4, n, k = 3, l = n - 3)$.

Note that, when $n = 9$, the settings (4) and (5) are identical to the settings (2) and (3) in Section 5.1.1 respectively. The evaluation budget (including initial evaluations) was set to $1,000$, the number of initial observations was set to 100, γ was set to 0.10, and n_c was set to 24 for MOTPE. On the other hand, the evaluation budget for NSGA-II was set to $10,000$. The remaining settings for NSGA-II were set to population_size $= 100$, mutation_prob $= 1/n$, crossover_prob $= 0.9$, and swapping_prob $= 0.5$.[6] For both methods, the initial 100 solutions were sampled using the Latin hypercube sampling (McKay et al.,

6. The documentation of NSGA-II parameters: https://optuna.readthedocs.io/en/v2.0.0/reference/multi_objective/generated/optuna.multi_objective.samplers.NSGAIIMultiObjectiveSampler.html.



(a) WFG1 (b) WFG2 (c) WFG3

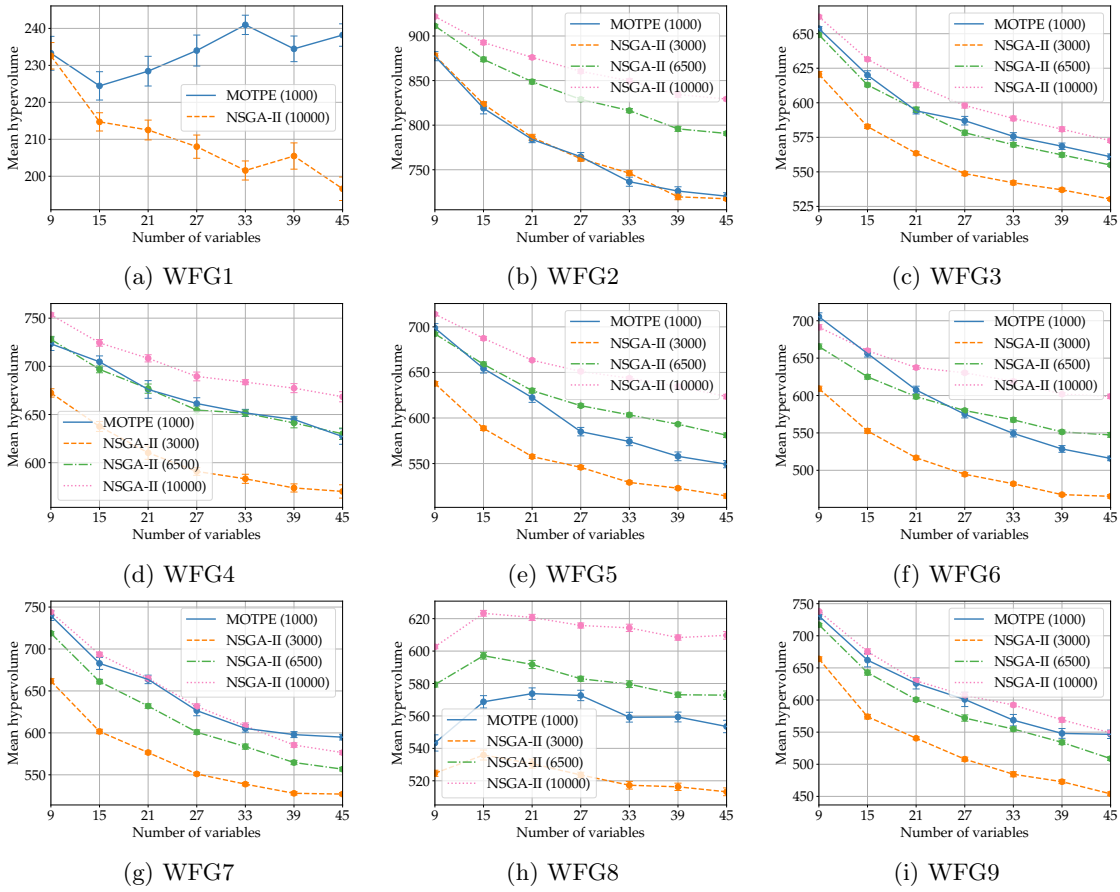
(d) WFG4 (e) WFG5 (f) WFG6

(g) WFG7 (h) WFG8 (i) WFG9

Figure 8: Mean hypervolume vs. Number of variables: $(m = 2, n, k = 1, l = n - 1)$ for $n = 9, 15, 21, 27, 33, 39, 45$. The number in parentheses after a method name indicates the number of evaluations, e.g., MOTPE (1000) shows the result of MOTPE after $1,000$ evaluations. The error bars represent the standard error.

1979). For each method and setting, 21 optimization runs were performed to get a mean hypervolume and a standard error. As a metric of performance, the hypervolume indicator with the reference point $(3, 5)$ (resp. $(3, 5, 7, 9)$) for $m = 2$ (resp. $m = 4$) was used as same as Section 5.1.1.

Figure 8 and Figure 9 show the obtained hypervolumes. Based on the results, when $m = 2$, MOTPE with $1,000$ evaluations achieves the same level of performance as NSGA-II with $1,500$–$2,500$ evaluations in many cases. When $m = 4$, MOTPE achieves the same level of performance as NSGA-II with $3,000$–$10,000$ evaluations. Thus, MOTPE is more effective than NSGA-II in medium-dimensional problems with $1,000$ evaluations. However, the results also indicate that the performance of MOTPE often degrades more than that of NSGA-II when the number of variables increases. One of the potential reasons for the performance degradation is that as the number of variables increases, it becomes more difficult to estimate the density of a promising region for each variable. Although MOTPE's



(a) WFG1 (b) WFG2 (c) WFG3

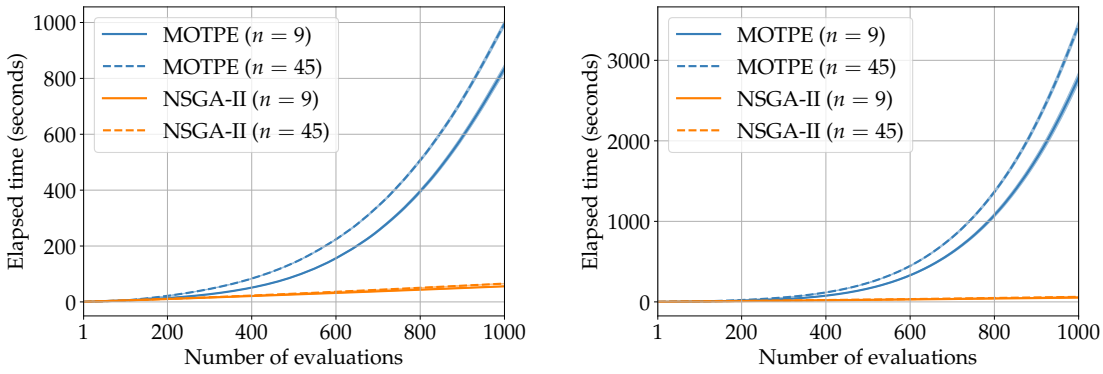
(d) WFG4 (e) WFG5 (f) WFG6

(g) WFG7 (h) WFG8 (i) WFG9

Figure 9: Mean hypervolume vs. Number of variables: $(m = 4, n, k = 3, l = n - 3)$ for $n = 9, 15, 21, 27, 33, 39, 45$. The number in parentheses after a method name indicates the number of evaluations. The error bars represent the standard error.

surrogate is more robust than GP in this respect based on the results in Section 5.1.1, it never means there is no limitation. In MOTPE, failure of the density estimation results in the sampling of poor candidates.

Then, the elapsed times of both methods averaged over all WFG problems are shown in Figure 10. Note that both methods were not parallelized in our experiments. The elapsed time of MOTPE is clearly a nonlinear function of the number of evaluations. This is because MOTPE maintains all of the previous observations and the computational complexity of operations such as the nondominated sorting is a nonlinear function of the number of vectors sorted. Therefore, we conclude that the maximum budget that MOTPE can handle within a reasonable time is at most a few thousand. On the other hand, the elapsed time of NSGA-II is a linear function of the number of evaluations because NSGA-II maintains a fixed size of population per generation.



(a) $m = 2$ (b) $m = 4$

Figure 10: Elapsed time vs. Number of evaluations averaged over all WFG problems. Shadings in (a) and (b) represent $\pm$ standard error.

Finally, we discuss when we should use MOTPE rather than NSGA-II. One possible scenario that MOTPE is more preferable to NSGA-II is financially expensive optimization. If our evaluation budget is limited because of financial constraints, we cannot increase our evaluation budget so much even if many parallel workers are available. Therefore, MOTPE is a better choice because it outperforms NSGA-II with the same evaluation budget. On the other hand, if many parallel workers are available and our evaluation budget can be increased, then parallelized NSGA-II seems to be more preferable. This is because NSGA-II can be synchronously parallelized without performance degradation up to its population size and the method with additional several thousand evaluations can outperform MOTPE.

5.1.3 CNN Design: Real-World Problem with Complex Search Space

Now, we compare MOTPE and other multiobjective Bayesian optimization algorithms on the multiobjective CNN design, which is an important real-world problem with a complex search space. Here, in addition to ParEGO, SMS-EGO, and PESMO, HyperMapper 2.0 (Nardi et al., 2019) is also used as a baseline. HyperMapper 2.0 employs random forests as its surrogate so that it generally handles integer and categorical variables better than GP-based methods.

The goal of our multiobjective CNN design problem is to design an accurate and fast to predict CNN for object recognition. Formally, the objectives of this problem are to minimize (1) the classification error rate for the CIFAR-10 dataset (Krizhevsky, 2009) and (2) the average elapsed time to make $1,000$ predictions. Therefore, our task is to find a set of Pareto optimal settings of several architecture parameters and hyperparameters in Table 2 for our architecture template shown in Table 3. The problem has a complex search space, which is a combination of real, integer, categorical, and conditional parameters. All parameters affect the first objective and the parameters except for Dropout rate, Stochastic Gradient Descent (SGD) learning rate, and SGD momentum affect the second objective. We remark that the first objective which requires training a CNN is computationally expensive to evaluate so that our evaluation budget cannot be large.

Parameter	Type	Search range
Number of blocks (N_B)	Integer	$[1, 3]$
Number of filters 1 (F_1)	Integer/conditional	$[16, 256]$
Number of filters 2 (F_2)	Integer/conditional	$[16, 256]$
Number of filters 3 (F_3)	Integer/conditional	$[16, 256]$
Batch normalization 1 (B_1)	Categorical/conditional	{False, True}
Batch normalization 2 (B_2)	Categorical/conditional	{False, True}
Batch normalization 3 (B_3)	Categorical/conditional	{False, True}
Pooling	Categorical	{Average, Max}
Dropout rate	Real	$[0.0, 0.9]$
Number of units (N_U)	Integer	$[16, 4096]$
SGD learning rate	Real	$[0.00001, 0.1]$
SGD momentum	Real	$[0.8, 1.0]$

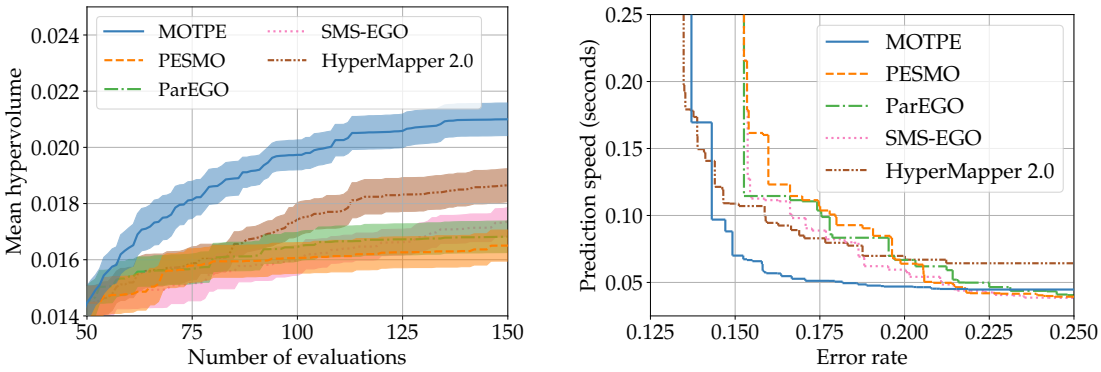
Table 2: Parameters of the multiobjective CNN design problem. The conditional parameter F_j and B_j are active iff $N_\mathrm{B} \geq j$ $(j = 1, 2, 3)$.

For all methods, we set the evaluation budget (including initial evaluations) to 150. The number of initial observations n_i was set to 50 following the single-objective CNN design using TPE (Bergstra et al., 2013). The initial solutions were sampled using random sampling because Latin hypercube sampling is not suitable for non-continuous variables. The method-specific settings for each method were set as follows. For MOTPE, we set $\gamma = 0.10$, $n_\mathrm{c} = 24$, the parameter scales for Number of units and SGD learning rate to log-uniform, and those for the rest of the numerical parameters to uniform. For Spearmint, we set likelihood = "GAUSSIAN" because the problem is noisy. Additionally, we marked Dropout rate, SGD learning rate, and SGD momentum as "to_ignore" for the second objective because our second objective does not depend on these parameters. The rest of the parameters for Spearmint were the same as in Table 1. For HyperMapper 2.0, we used the default settings of the framework.[7] In GP-based methods, categorical parameters were encoded to integers (e.g., False = 0 and True = 1). In methods other than MOTPE, conditional parameters were always sampled and ignored at evaluation if they were inactive. The CNNs were implemented in the tf.keras (Tensorflow version 2.2.0) library and trained using the SGD optimizer with a batch size of 32 during 50 epochs.[8] The error rate was measured on a set of $10,000$ images extracted from the training set. The rest of the training data, $40,000$ images, were used for training. For each method and setting, 12 optimization runs were performed to get a mean hypervolume and a standard error. As a metric of performance, we used the hypervolume indicator with the reference point $(0.25, 0.25)$ because we wanted accurate and fast to predict models. Especially, we considered that a fairly inaccurate model is practically useless even if it is fast.

7. The default settings of HyperMapper 2.0 is available at: https://github.com/luinardi/hypermapper/blob/ab618023722047c8d6b94a93f5438770c0b8d103/hypermapper/schema.json.
8. The code is available at https://doi.org/10.5281/zenodo.6258358.

Component		Output size	Description
Input		$32 \times 32 \times 3$	
(Block 1)	Convolution	$32 \times 32 \times F_1$	Kernel size: 3×3, Stride: 1, Padding: same, Activation: Rectified Linear Unit (ReLU)
	Convolution	$32 \times 32 \times F_1$	Kernel size: 3×3, Stride: 1, Padding: same, Activation: ReLU
	Pooling	$16 \times 16 \times F_1$	Pool size: 2×2
	(Batch normalization 1)	$16 \times 16 \times F_1$	
	Dropout	$16 \times 16 \times F_1$	
(Block 2)	Convolution	$16 \times 16 \times F_2$	Kernel size: 3×3, Stride: 1, Padding: same, Activation: ReLU
	Convolution	$16 \times 16 \times F_2$	Kernel size: 3×3, Stride: 1, Padding: same, Activation: ReLU
	Pooling	$8 \times 8 \times F_2$	Pool size: 2×2
	(Batch normalization 2)	$8 \times 8 \times F_2$	
	Dropout	$8 \times 8 \times F_2$	
(Block 3)	Convolution	$8 \times 8 \times F_3$	Kernel size: 3×3, Stride: 1, Padding: same, Activation: ReLU
	Convolution	$8 \times 8 \times F_3$	Kernel size: 3×3, Stride: 1, Padding: same, Activation: ReLU
	Pooling	$4 \times 4 \times F_3$	Pool size: 2×2
	(Batch normalization 3)	$4 \times 4 \times F_3$	
	Dropout	$4 \times 4 \times F_3$	
Flatten		$(32/2^{N_\mathrm{B}})^2 F_{N_\mathrm{B}}$	
Fully-connected		N_U	Activation: ReLU
Dropout		N_U	
Fully-connected		10	Activation: softmax

Table 3: Architecture template for the multiobjective CNN design problem. Note that the component (Block j) exists iff $N_\mathrm{B} \geq j$ and the component (Batch normalization j) exists iff $N_\mathrm{B} \geq j \wedge B_j$ for $j = 1, 2, 3$.



(a) Mean hypervolume ± standard error.

(b) 50%-attainment surface.

Figure 11: Results of the multiobjective CNN design. Shadings in (a) represent ± standard error.

Figure 11 shows the results of the experiment. The results of initialization are removed from Figure 11a because they are the same for all methods. MOTPE outperforms all of the four baseline methods in achieved hypervolume values (Figure 11a). Figure 11b shows that MOTPE can find a variety of models with preferable trade-offs between error rate and prediction speed. As for the baseline methods, the GP-based methods, i.e., ParEGO, SMS-EGO, and PESMO, generally find fast to predict models but those are relatively inaccurate whereas HyperMapper 2.0 finds accurate models but those are relatively slow. Unlike the WFG benchmark problems, the landscapes of the objective functions are unknown for the multiobjective CNN design, so it is difficult to have a detailed discussion. However, our results indicate that MOTPE handles the complex search space better than the baseline methods and has a sufficient ability for practical use.

5.2 Investigations of MOTPE

Here, we conduct two additional experiments: an investigation of the influence of quantile γ and an investigation of the effectiveness of parallelization.

5.2.1 Influence of Quantile γ

The quantile parameter γ of MOTPE has a significant impact on constructing its surrogate. It is the most important parameter relative to optimization performance. Then, we empirically investigated the effects of γ on MOTPE's behavior and identify the optimal setting for γ.

To investigate the influence of γ, MOTPE was tested on the WFG benchmark problems with different γ settings. There were 21 settings in the experiment: $\gamma = 0.01, 0.05, 0.10, \ldots, 0.95, 0.99$. We prepared four problem settings that were the three settings in Section 5.1.1, i.e., $(m = 2, n = 3, k = 1, l = 2)$, $(m = 2, n = 9, k = 1, l = 8)$, and $(m = 4, n = 9, k = 3, l = 6)$, and a new one $(m = 2, n = 9, k = 3, l = 6)$. The evaluation budget was set to 250 and the number of candidates n_c was set to 24 as same as

Section 5.1.1. To test the effect of the number of initial samples n_{i}, we prepared the two settings: $n_{\mathrm{i}} = 32$ and $n_{\mathrm{i}} = 98$ for each problem. For each setting, 21 optimization runs were performed, using the hypervolume indicator as the performance metric with the reference point $(3, 5)$ (resp. $(3, 5, 7, 9)$) for $m = 2$ (resp. $m = 4$).

Figure 12 shows the mean hypervolumes for each γ setting. The observations made as the results are as follows.

(1) There seem to be roughly two types of the problems: the problems with the relatively large best γ (e.g., WFG1 and WFG4) and the problems with the relatively small best γ (e.g., WFG9).

(2) The best γ for $n = 3$ tends to be greater than that for $n = 9$.

(3) The best γ for $k = 3$ tends to be greater than that for $k = 1$.

(4) The best γ for $n_{\mathrm{i}} = 32$ tends to be greater than that for $n_{\mathrm{i}} = 98$.

(5) The best γ for $m = 4$ tends to be greater than that for $m = 2$.

(6) Extremely small or large γ always results in poor performance.

(7) The relationship between the mean hypervolume and γ is roughly mountain-shape.

Based on the results and the observations, our empirical recommendation is $\gamma = 0.10$. This setting was the winner in 17 out of 72 experiments in Figure 12. In summary, we conclude that γ controls the balance of the pressure to converge and to make diversity. Our results consistently show that a smaller γ has a stronger pressure to converge whereas a larger γ has a stronger pressure to make diversity. A detailed discussion for each observation is provided in the following.

Let us have a discussion for each observation to understand the effect of γ. Regarding the observation (1), the key point is the convergence difficulty towards the ideal point of each problem. WFG1 is extremely biased so that convergence towards the ideal point is hard in this problem. Therefore, the achieved hypervolume is nearly determined by the diversity of observations. In Figure 13a, the results show that larger γ achieves better diversity. On the other hand, WFG4 is a very easy-to-converge problem. In Figure 13b, $\gamma = 0.10$, 0.25, and 0.40 achieve almost the same level of good convergence for WFG4. WFG5 might be also considered as this type of problem. WFG9 is a problem that is neither too easy nor too difficult to converge. In Figure 13c, the results show that smaller γ achieves much faster convergence. WFG2, WFG3, and WFG6–8 might be also categorized in this type of problem.

Regarding the observation (2), the WFG problems with $n = 9$ are generally more difficult to converge than those with $n = 3$ because the search space of the former is larger. Therefore, when $n = 9$, achieving better hypervolume requires a stronger pressure to converge than when $n = 3$. In Figure 14, it can be confirmed that $\gamma = 0.10$ converges better than $\gamma = 0.25$, especially when $n = 9$.

Regarding the observation (3), increasing the number of position-related parameters k of the WFG problems generally makes the problems more difficult in terms of diversity. Therefore, when $k = 3$, achieving better hypervolume requires a stronger pressure to make

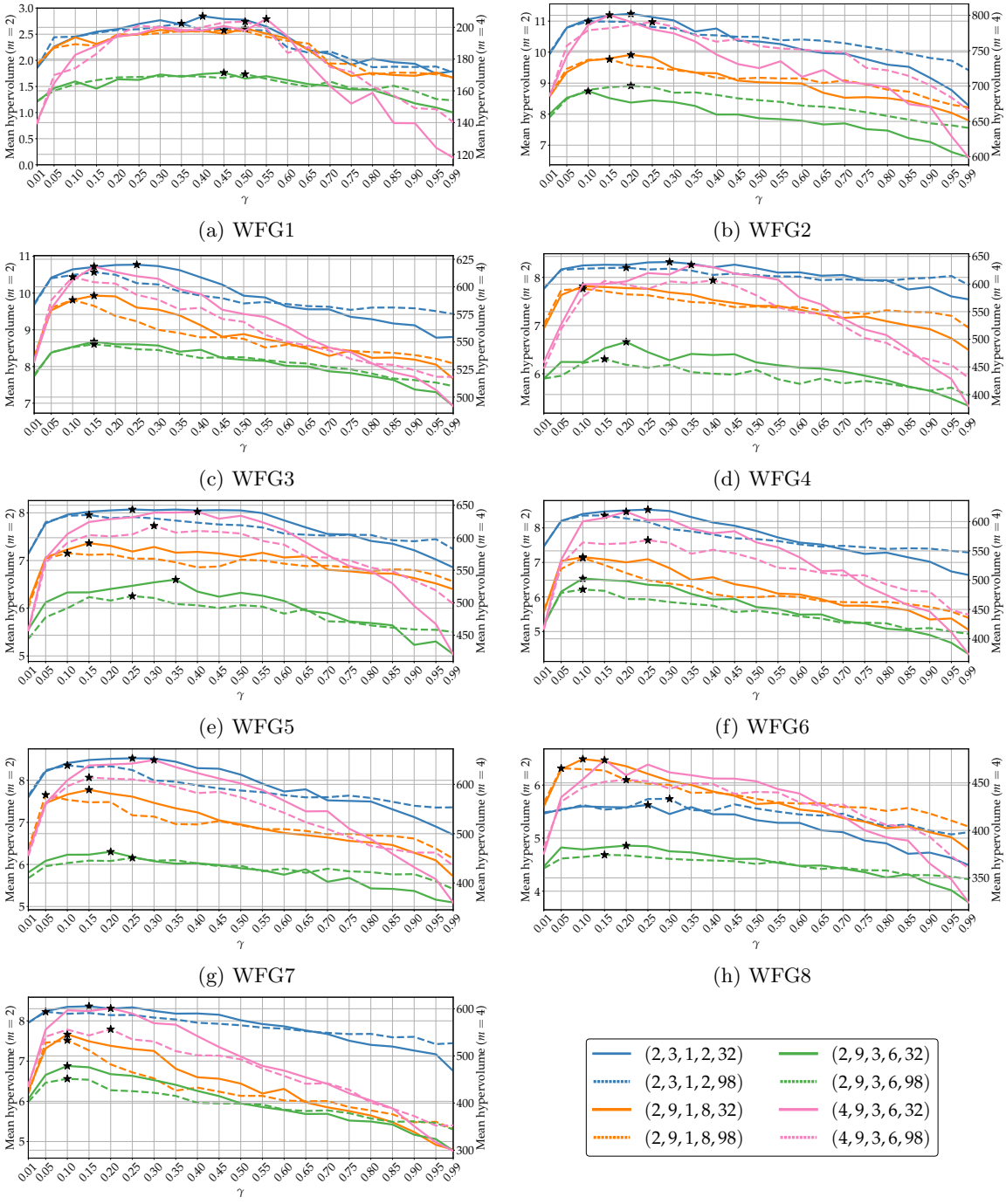
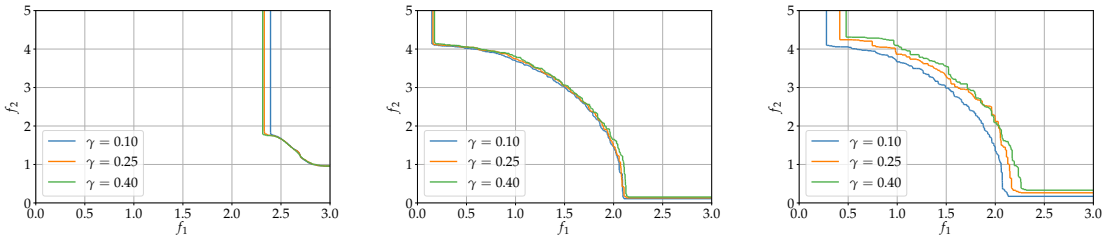
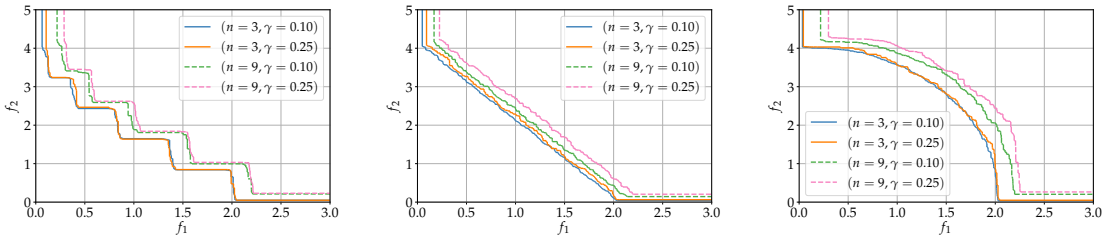


Figure 12: Mean hypervolume vs. γ setting. Legends in the lower right mean (m, n, k, l, n_{i}) where m is the number of objectives, n is the number of variables, k and l are the working parameters of the WFG problems, and n_{i} is the number of initial samples. The best value on each experiment is marked with $\star$.



(a) WFG1 (b) WFG4 (c) WFG9

Figure 13: 50%-attainment surface: $(m = 2, n = 9, k = 1, l = 8, n_{\mathrm{i}} = 98)$.



(a) WFG2 (b) WFG3 (c) WFG6

Figure 14: 50%-attainment surface: $(m = 2, k = 1, l = 8, n_{\mathrm{i}} = 98)$.

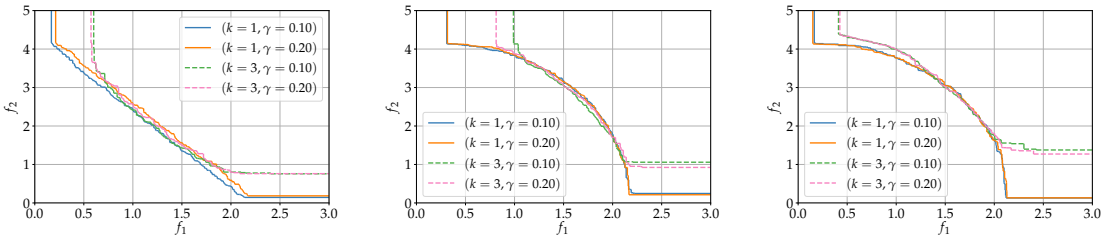
diversity than when $k = 1$. In Figure 15, there are three examples that a smaller γ converges better than a larger one while both γ settings achieve the same level of diversity when $k = 1$. On the other hand, a larger γ achieves better diversity than a smaller one when $k = 3$, and the preferable γ is swapped compared to the case of $k = 1$.

Regarding the observation (4), the key point is the number of iterations for MOTPE. When we have the fixed evaluation budget 250, the number of iterations is equal to $250 - n_{\mathrm{i}}$. Therefore, it generally needs to converge faster when $n_{\mathrm{i}} = 98$ than when $n_{\mathrm{i}} = 32$. In Figure 16, there are three examples that a larger γ achieves better diversity than a smaller one while both γ settings achieve the same level of convergence when $n_{\mathrm{i}} = 32$. On the other hand, a smaller γ converges better than a larger one when $n_{\mathrm{i}} = 98$, and the preferable γ is swapped compared to the case of $n_{\mathrm{i}} = 32$.

Regarding the observation (5), the key point is the ratio of nondominated observations. In general, the larger m is, the greater the ratio of nondominated observations is because the higher the dimension is, the more difficult it is to satisfy the condition $\forall i : y_i \leq y'_i$ for two vectors $\boldsymbol{y}$ and $\boldsymbol{y}'$. Therefore, we consider that when $m = 4$, the ratio of good observations, γ, is preferred to be large compared to when $m = 2$.

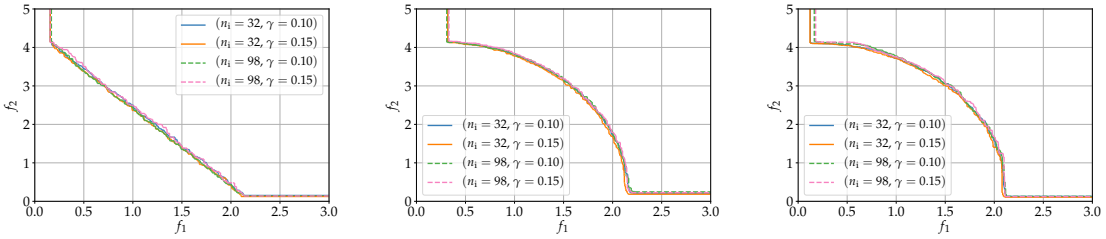
Observation (6) gives us an empirical proper range for γ. As shown in Figure 12, γ settings out of the proper range always degrade the performance of MOTPE. This information is useful for tuning of γ.

Observation (7) indicates that γ is relatively robust to the performance of MOTPE. Therefore, fortunately, tuning γ seems to be not so difficult.



(a) WFG3 (b) WFG5 (c) WFG7

Figure 15: 50%-attainment surface: $(m = 2, n = 9, l = n - k, n_{\mathrm{i}} = 98)$.



(a) WFG3 (b) WFG5 (c) WFG7

Figure 16: 50%-attainment surface: $(m = 2, n = 9, k = 1, l = 8)$.

5.2.2 Effectiveness of Parallelization

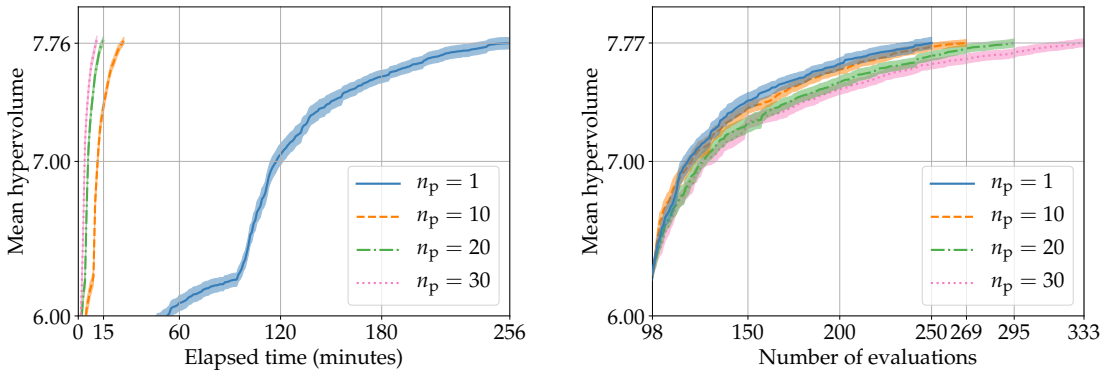
In this subsection, we demonstrate the effectiveness of the asynchronous parallel MOTPE algorithm introduced in Section 3.3. Parallelization is vital to solving real-world problems in practice.

To test the effectiveness of parallelization, we solved the WFG benchmark problems with the setting $(m = 2, n = 9, k = 1, l = 8)$ by using Algorithm 5 with the number of workers $n_{\mathrm{p}} = 1, 10, 20, 30$. We set the evaluation budget to 250 for $n_{\mathrm{p}} = 1$, which means no parallelization, and more than 250 for $n_{\mathrm{p}} = 10, 20, 30$ to investigate the degradation of the sample efficiency due to parallelization. When evaluating the objective, we set the algorithm to sleep $s \overset{iid}{\sim} \mathcal{N}(\mu = 60, \sigma^2 = 15^2)$ seconds to simulate the situation that objectives are computationally expensive. The rest of the method and the problem settings were the same as in Section 5.1.1. For each method and setting, 21 optimization runs were performed to get a mean hypervolume and a standard error.

Figure 17 shows the elapsed time and the number of evaluations to achieve the same level of hypervolume as $n_{\mathrm{p}} = 1$ on WFG4 as a representative. The results of the initialization step are omitted in Figure 17b. Table 4 shows the results of all WFG problems.

We find that the elapsed time to achieve the baseline hypervolume was drastically reduced by parallelization for all problems. Speedups for each setting were $257.0/27.9 \approx 9.21$, $257.0/17.7 \approx 14.5$, and $257.0/13.1 \approx 19.6$ respectively. Parallelization efficiencies for each setting were $257.0/(27.9 \times 10) \approx 0.92$, $257.0/(17.7 \times 20) \approx 0.73$, and $257.0/(13.1 \times 30) \approx 0.65$ respectively. This effectiveness is based on the asynchrony of the parallelization in MOTPE.

As for the number of evaluations, we find that the more workers we use, the more observations are needed to achieve the baseline. Therefore, the sample efficiency of MOTPE



(a) Mean hypervolume vs. Number Elapsed time.

(b) Mean hypervolume vs. Number of evaluations.

Figure 17: Parallelization: WFG4. Shadings in (a) and (b) represent ± standard error.

Problem	$n_p = 1$	10	20	30
WFG1	257	24	19	12
WFG2	258	28	17	13
WFG3	258	28	17	15
WFG4	256	27	15	11
WFG5	257	25	15	11
WFG6	256	34	21	16
WFG7	258	28	16	12
WFG8	256	26	17	12
WFG9	257	31	22	16
Mean	257.0	27.9	17.7	13.1

(a) Elapsed time (minutes).

Problem	$n_p = 1$	10	20	30
WFG1	250	247	389	348
WFG2	250	275	341	401
WFG3	250	279	350	448
WFG4	250	269	295	333
WFG5	250	246	291	327
WFG6	250	343	421	480
WFG7	250	277	319	367
WFG8	250	258	347	352
WFG9	250	307	434	470
Mean	250.0	277.9	354.1	391.8

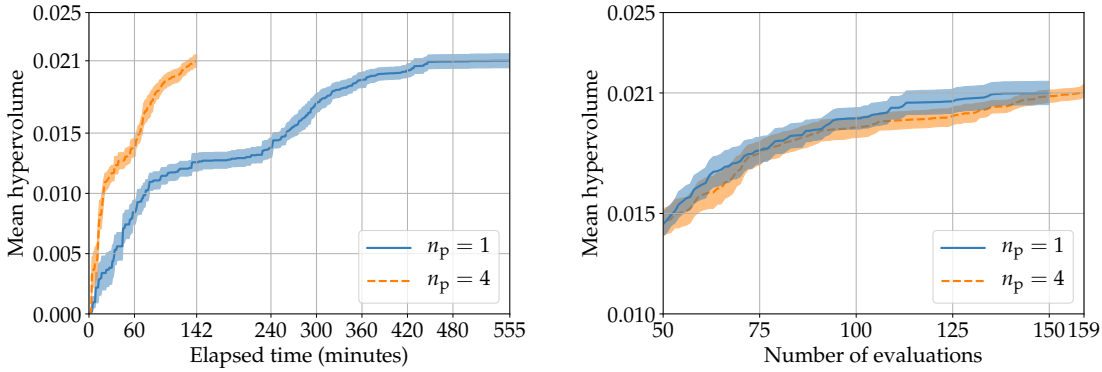
(b) Number of evaluations.

Table 4: Elapsed time and the number of evaluations to achieve the same level of hypervolume as $n_p = 1$.

decreases by parallelization. However, we consider that additional tens to a few hundreds of evaluations speed up the algorithm about 10–20 times is practical enough.

Finally, we solved the multiobjective CNN design problem described in Section 5.1.3 using MOTPE with four workers ($n_p = 4$) to demonstrate its practicality in the real-world problem. Although $n_p = 4$ is relatively small, this is a realistic setting as it is not easy to prepare dozens of graphics processing units. The remaining method/problem settings were the same as applied in Section 5.1.3. We ran MOTPE ($n_p = 4$) to reach the same level of hypervolume as MOTPE ($n_p = 1$) shown in Figure 11a.

Figure 18 shows the results of the experiment. We find that parallelization drastically speeds up MOTPE. The speedup was $142/555 \approx 0.26$. The average time, the minimum time, and the maximum time required for an evaluation among all evaluations were about 765.4 ± 1085.5 ($\pm$ indicates the standard deviation), 69.4, and 4391.7 seconds respectively



(a) Mean hypervolume vs. Elapsed time.

(b) Mean hypervolume vs. Number of evaluations.

Figure 18: Parallelization: Multiobjective CNN design. Shadings in (a) and (b) represent ± standard error.

when $n_p = 4$. This fact indicates the importance of asynchronous parallelization without wait time in practice. In the end, we conclude that asynchronously parallelized MOTPE is quite effective and practical.

6. Concluding Remarks

In this study, we have introduced MOTPE, which is a practical multiobjective Bayesian optimization algorithm. MOTPE has been designed to solve problems having a complex search space with a limited number of evaluations. The method has also been designed to be scalable compared to standard GP-based methods and effectively parallelized in an asynchronous manner. Our numerical results have demonstrated that MOTPE can solve a variety of problems and has a great ability for practical use. It has also been shown that MOTPE is useful to solve problems with the medium dimension and medium budget setting. In the investigations of MOTPE, we have found that MOTPE's γ parameter controls the balance of the pressure to converge and to make diversity. We have also validated the effectiveness of asynchronous parallel MOTPE.

There are still some issues that MOTPE has not addressed. One, for instance, is constrained multiobjective optimization (Feliot et al., 2017; Garrido-Merchán & Hernández-Lobato, 2019) and another is multiobjective multifidelity optimization (Belakaria, Deshwal, & Doppa, 2020). Falkner, Klein, and Hutter (2018) proposed a multifidelity single-objective Bayesian optimization method called BOHB based on TPE and Hyperband (Li, Jamieson, DeSalvo, Rostamizadeh, & Talwalkar, 2017). Therefore, the idea of multifidelity MOTPE seems to be pragmatic. One more example is solving extremely biased problems. Based on our results in Section 5.1.1, all tested algorithms could not solve the extremely biased WFG1 problem well. There may be an extremely biased real-world problem although the multiobjective CNN design problem in Section 5.1.3 was not so biased. Some nontrivial algorithm improvements are needed to solve such extremely biased problems. These can be considered as our future directions.

Acknowledgments

We are grateful to the anonymous reviewers for their helpful comments and suggestions. We are also grateful to Hideaki Imamura in Preferred Networks, Inc. for helpful discussions. This paper is based on the results obtained from a project commissioned by the New Energy and Industrial Technology Development Organization (NEDO). Computational resource of AI Bridging Cloud Infrastructure (ABCI) provided by National Institute of Advanced Industrial Science and Technology (AIST) was used.

Appendices

Appendix A provides the derivation of Probability of Hypervolume Improvement (PHVI). Appendix B investigates whether enhancing the ability to sample boundary points improves the performance of MOTPE or not.

Appendix A. Probability of Hypervolume Improvement

Here we introduce an acquisition function called PHVI for multiobjective optimization. PHVI for Y^* and x_i is defined as follows:

$$\mathrm{PHVI}_{Y^*}(x_i) \coloneqq p(\boldsymbol{y} \in R \mid x_i) \tag{12}$$

where $R = \{\boldsymbol{y} \mid (\boldsymbol{y} \prec Y^*) \vee (\boldsymbol{y} \parallel Y^*)\}$. This PHVI criterion is a natural extension of Probability of Improvement (PI) (Kushner, 1964), which is a well-known criterion for single-objective Bayesian optimization.

Then, we show that the two acquisition functions of PHVI and EHVI work equivalent in MOTPE. As same as Equation (10), based on Equation (8), $\gamma = p(\boldsymbol{y} \in R)$, and $p(x_i) = \int p(x_i \mid \boldsymbol{y}) p(\boldsymbol{y}) d\boldsymbol{y} = \gamma l(x_i) + (1-\gamma) g(x_i)$; therefore, $p(\boldsymbol{y} \in R \mid x_i)$ is calculated as below:

$$\begin{aligned} p(\boldsymbol{y} \in R \mid x_i) &= \int_R p(\boldsymbol{y} \mid x_i) d\boldsymbol{y} \\ &= \int_R \frac{p(x_i \mid \boldsymbol{y}) p(\boldsymbol{y})}{p(x_i)} d\boldsymbol{y} \\ &= \frac{\gamma l(x_i)}{\gamma l(x_i) + (1-\gamma) g(x_i)} \\ &= \frac{\gamma}{\gamma + (1-\gamma)\frac{g(x_i)}{l(x_i)}} \\ &\propto \left(\gamma + (1-\gamma)\frac{g(x_i)}{l(x_i)}\right)^{-1}. \end{aligned} \tag{13}$$

Thus, based on Equation (11) and Equation (13),

$$\begin{aligned} \operatorname*{argmax}_{x_i \in X_i} \mathrm{PHVI}_{Y^*}(x_i) &= \operatorname*{argmax}_{x_i \in X_i} \frac{l(x_i)}{g(x_i)} \\ &= \operatorname*{argmax}_{x_i \in X_i} \mathrm{EHVI}_{Y^*}(x_i) \end{aligned} \tag{14}$$

holds in MOTPE.

Appendix B. Enhancing Ability to Sample Boundary Points

In Section 5.1.1, we have discussed that MOTPE achieved relatively poor results on WFG5 because the algorithm has less ability to sample boundary points in the search space. Here we confirm whether enhancing this ability actually improves the performance of MOTPE or not.

To enhance the ability, we modified the sampling procedure of MOTPE as follows. First, (non-truncated) Gaussian was used to estimate densities and to sample candidate values instead of truncated Gaussian used in the original. Second, if the sampled value was out of bounds, clipping to the boundary value was applied.

We tested and compared the modified version of MOTPE and the original MOTPE on the WFG problems with the experimental settings used in Section 5.1.1. For each method, 51 optimization runs were performed to get a mean hypervolume and a standard error. The Wilcoxon rank sum test was conducted at the 0.05 significance level to verify the performance improvement.

Table A1 shows the results of the original and the modified MOTPEs after 250 evaluations. The results show that enhancing the ability actually improves the performance of MOTPE on WFG5 to some degree. The modified MOTPE achieved 8.01 ± 0.01, 7.30 ± 0.05, and 602.01 ± 3.37 on WFG5 $(2, 3, 1, 2)$, $(2, 9, 1, 8)$, and $(4, 9, 3, 6)$ respectively. These results are comparable to or better than the results achieved by PESMO that are 7.95 ± 0.02, 7.31 ± 0.08, and 505.58 ± 7.92 in Section 5.1.1. There are also other improved cases such as WFG4 $(2, 3, 1, 2)$ and WFG7 $(4, 9, 3, 6)$. However, the modification also caused performance degradation on WFG1 $(2, 3, 1, 2)$ and WFG1 $(2, 9, 1, 8)$.

Then, we briefly investigate how the behavior of MOTPE changed by the modification. Figure A1 shows the nondominated vectors found after 250 evaluations by the original and the modified MOTPEs on WFG5 $(2, 9, 1, 8)$ and WFG1 $(2, 9, 1, 8)$ by an optimization run as examples. The figure also shows the solutions (i.e., the values of $x_1, \ldots, x_9$) for some objective vectors. In Figure A1a, we confirm that the modified MOTPE could sample solutions on the boundary of the search space as expected. These solutions are certainly better than the solutions that are not on the boundary found by the original MOTPE. As a result, the modified MOTPE achieved better results than the original MOTPE on WFG5 $(2, 9, 1, 8)$. In Figure A1b, we find that the modified MOTPE found the two solutions on the boundary with the best f_1 value and the best f_2 value among all solutions sampled. However, we also find that the modified MOTPE is worse than the original MOTPE in the ability to find solutions achieving moderate trade-offs between f_1 and f_2. Based on the figure and our results, we find that solutions with the moderate trade-offs between the two objectives have a small but positive x_1 value, i.e., $x_1 \in (0, \epsilon)$ for small $\epsilon \in \mathbb{R}$, such as $x_1 = 0.0002\ldots$ on WFG1 $(2, 9, 1, 8)$ as shown in Figure A1b. Unfortunately, our modification that makes MOTPE easy to sample boundary values seems to make it difficult to sample values near the boundary but not on the boundary instead and this results in the poor performance on WFG1 $(2, 9, 1, 8)$.

In the end, we conclude that enhancing the ability to sample boundary points is sometimes helpful but we need to be aware that small differences in the search space can result in large differences in the objective space when a target problem is extremely biased.

Problem	MOTPE (original)	MOTPE (modified)
WFG1	$\mathbf{2.47 \pm 0.03}$	1.57 ± 0.02
WFG2	11.08 ± 0.01	$\mathbf{11.11 \pm 0.01}$
WFG3	10.64 ± 0.01	$\mathbf{10.66 \pm 0.01}$
WFG4	8.25 ± 0.01	$\mathbf{8.28 \pm 0.01}$
WFG5	7.96 ± 0.01	$\mathbf{8.01 \pm 0.01}$
WFG6	8.40 ± 0.01	8.41 ± 0.01
WFG7	8.41 ± 0.00	$\mathbf{8.44 \pm 0.00}$
WFG8	5.60 ± 0.04	5.67 ± 0.04
WFG9	8.34 ± 0.01	8.35 ± 0.01

(a) Setting: $(m = 2, n = 3, k = 1, l = 2)$.

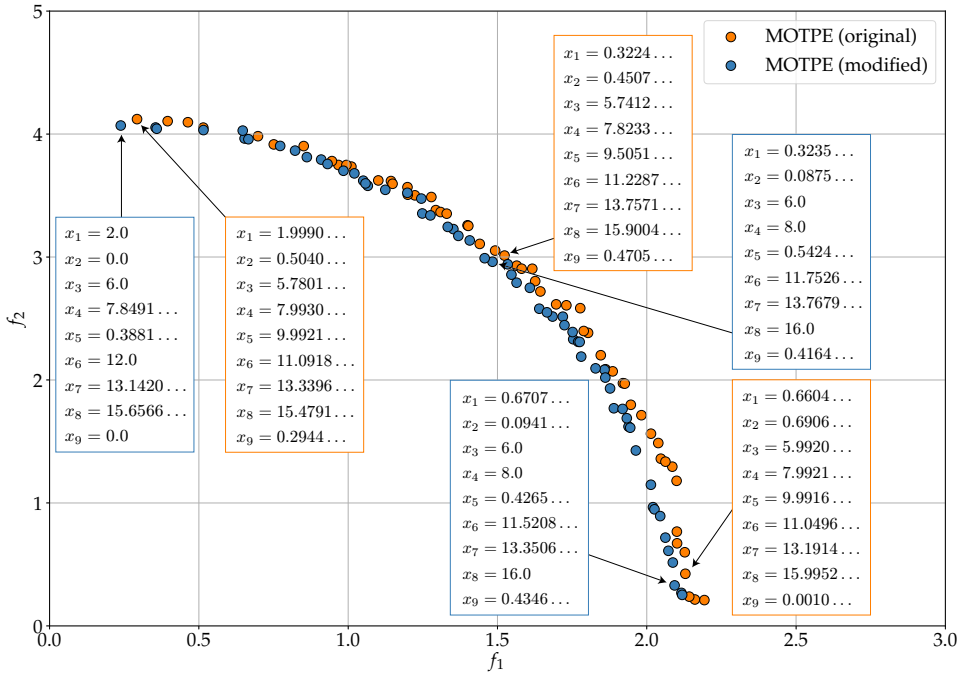
Problem	MOTPE (original)	MOTPE (modified)
WFG1	$\mathbf{2.34 \pm 0.03}$	1.50 ± 0.03
WFG2	9.70 ± 0.06	9.79 ± 0.06
WFG3	9.75 ± 0.04	9.81 ± 0.04
WFG4	7.78 ± 0.02	7.79 ± 0.02
WFG5	7.16 ± 0.04	$\mathbf{7.30 \pm 0.05}$
WFG6	7.10 ± 0.05	7.05 ± 0.05
WFG7	7.66 ± 0.05	7.64 ± 0.05
WFG8	6.31 ± 0.03	6.28 ± 0.03
WFG9	7.38 ± 0.07	7.40 ± 0.07

(b) Setting: $(m = 2, n = 9, k = 1, l = 8)$.

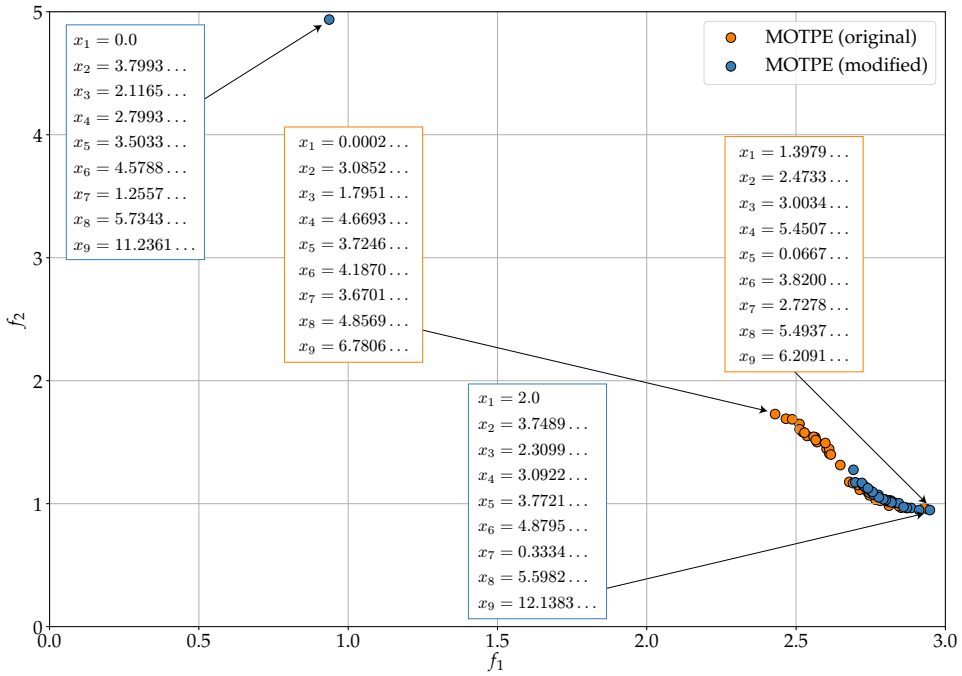
Problem	MOTPE (original)	MOTPE (modified)
WFG1	178.76 ± 2.52	$\mathbf{195.34 \pm 4.27}$
WFG2	780.10 ± 3.66	778.90 ± 3.81
WFG3	608.29 ± 1.72	608.80 ± 1.92
WFG4	593.37 ± 6.77	$\mathbf{616.06 \pm 7.34}$
WFG5	590.04 ± 3.24	$\mathbf{602.01 \pm 3.37}$
WFG6	563.93 ± 4.24	565.32 ± 4.04
WFG7	602.53 ± 5.95	$\mathbf{616.02 \pm 5.47}$
WFG8	442.84 ± 3.15	447.10 ± 2.87
WFG9	562.43 ± 6.75	575.83 ± 7.16

(c) Setting: $(m = 4, n = 9, k = 3, l = 6)$.

Table A1: Mean hypervolume $\pm$ standard error. Bold means that the performance difference is statistically significant.



(a) WFG5 $(2, 9, 1, 8)$.



(b) WFG1 $(2, 9, 1, 8)$.

Figure A1: Scatter plots of the nondominated vectors found. Each set of values $\{x_1, \ldots, x_9\}$ is the solution for the corresponding point. Each value is shown up to the fourth decimal place for readability.

References

Abdolshah, M., Shilton, A., Rana, S., Gupta, S., & Venkatesh, S. (2019). Multi-objective Bayesian optimisation with preferences over objectives. In *Advances in Neural Information Processing Systems*, pp. 12235–12245.

Akiba, T., Sano, S., Yanase, T., Ohta, T., & Koyama, M. (2019). Optuna: A next-generation hyperparameter optimization framework. In *ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 2623–2631.

Archetti, F., & Candelieri, A. (2019). *Bayesian Optimization and Data Science*. Springer.

Azimi, J., Fern, A., & Fern, X. Z. (2010). Batch Bayesian optimization via simulation matching. In *Advances in Neural Information Processing Systems*, pp. 109–117.

Bader, J., & Zitzler, E. (2011). HypE: An algorithm for fast hypervolume-based many-objective optimization. *Evolutionary Computation*, *19*(1), 45–76.

Belakaria, S., Deshwal, A., & Doppa, J. R. (2019). Max-value entropy search for multi-objective Bayesian optimization. In *Advances in Neural Information Processing Systems*, pp. 7825–7835.

Belakaria, S., Deshwal, A., & Doppa, J. R. (2020). Multi-fidelity multi-objective Bayesian optimization: An output space entropy search approach. In *AAAI*, pp. 10035–10043.

Bergstra, J., Bardenet, R., Bengio, Y., & Kégl, B. (2011). Algorithms for hyper-parameter optimization. In *Advances in Neural Information Processing Systems*, pp. 2546–2554.

Bergstra, J., Komer, B., Eliasmith, C., Yamins, D., & Cox, D. D. (2015). Hyperopt: A Python library for model selection and hyperparameter optimization. *Computational Science & Discovery*, *8*(1), 014008.

Bergstra, J., Yamins, D., & Cox, D. (2013). Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. In *International Conference on Machine Learning*, pp. 115–123.

Bradstreet, L., While, L., & Barone, L. (2007). Incrementally maximising hypervolume for selection in multi-objective evolutionary algorithms. In *2007 IEEE Congress on Evolutionary Computation*, pp. 3203–3210.

Campigotto, P., Passerini, A., & Battiti, R. (2013). Active learning of Pareto fronts. *IEEE Transactions on Neural Networks and Learning Systems*, *25*(3), 506–519.

Cheng, R., Jin, Y., Olhofer, M., & Sendhoff, B. (2016). A reference vector guided evolutionary algorithm for many-objective optimization. *IEEE Transactions on Evolutionary Computation*, *20*(5), 773–791.

Chugh, T., Jin, Y., Miettinen, K., Hakanen, J., & Sindhya, K. (2018). A surrogate-assisted reference vector guided evolutionary algorithm for computationally expensive many-objective optimization. *IEEE Transactions on Evolutionary Computation*, *22*(1), 129–142.

Chugh, T., Sindhya, K., Hakanen, J., & Miettinen, K. (2019). A survey on handling computationally expensive multiobjective optimization problems with evolutionary algorithms. *Soft Computing*, *23*(9), 3137–3166.

Chugh, T., Sindhya, K., Miettinen, K., Jin, Y., Kratky, T., & Makkonen, P. (2017). Surrogate-assisted evolutionary multiobjective shape optimization of an air intake ventilation system. In *IEEE Congress on Evolutionary Computation*, pp. 1541–1548.

Chugh, T., Sun, C., Wang, H., & Jin, Y. (2020). Surrogate-assisted evolutionary optimization of large problems. In *High-Performance Simulation-Based Optimization*, pp. 165–187. Springer.

Costa, M., & Minisci, E. (2003). MOPED: A multi-objective Parzen-based estimation of distribution algorithm for continuous problems. In *International Conference on Evolutionary Multi-Criterion Optimization*, pp. 282–294.

Cuesta-Ramirez, J., Riche, R. L., Roustant, O., Perrin, G., Durantin, C., & Gliere, A. (2021). A comparison of mixed-variables Bayesian optimization approaches..

Deb, K., Pratap, A., Agarwal, S., & Meyarivan, T. (2002). A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, *6*(2), 182–197.

Emmerich, M. T., Deutz, A. H., & Klinkenberg, J. W. (2011). Hypervolume-based expected improvement: Monotonicity properties and exact computation. In *2011 IEEE Congress of Evolutionary Computation (CEC)*, pp. 2147–2154. IEEE.

Emmerich, M. T., Giannakoglou, K. C., & Naujoks, B. (2006). Single- and multiobjective evolutionary optimization assisted by Gaussian random field metamodels. *IEEE Transactions on Evolutionary Computation*, *10*(4), 421–439.

Falkner, S., Klein, A., & Hutter, F. (2018). BOHB: Robust and efficient hyperparameter optimization at scale. In *International Conference on Machine Learning*, pp. 1437–1446.

Feliot, P., Bect, J., & Vazquez, E. (2017). A Bayesian approach to constrained single- and multi-objective optimization. *Journal of Global Optimization*, *67*(1-2), 97–133.

Fleischer, M. (2003). The measure of Pareto optima applications to multi-objective metaheuristics. In *International Conference on Evolutionary Multi-Criterion Optimization*, pp. 519–533.

Garrido-Merchán, E. C., & Hernández-Lobato, D. (2019). Predictive entropy search for multi-objective Bayesian optimization with constraints. *Neurocomputing*, *361*, 50–68.

González, J., Dai, Z., Hennig, P., & Lawrence, N. (2016). Batch Bayesian optimization via local penalization. In *International Conference on Artificial Intelligence and Statistics*, pp. 648–657.

Guerreiro, A. P., Fonseca, C. M., & Paquete, L. (2016). Greedy hypervolume subset selection in low dimensions. *Evolutionary Computation*, *24*(3), 521–544.

Guo, D., Jin, Y., Ding, J., & Chai, T. (2018). Heterogeneous ensemble-based infill criterion for evolutionary multiobjective optimization of expensive problems. *IEEE Transactions on Cybernetics*, *49*(3), 1012–1025.

Hernández-Lobato, D., Hernández-Lobato, J. M., Shah, A., & Adams, R. (2016). Predictive entropy search for multi-objective Bayesian optimization. In *International Conference on Machine Learning*, pp. 1492–1501.

Hernández-Lobato, J. M., Hoffman, M. W., & Ghahramani, Z. (2014). Predictive entropy search for efficient global optimization of black-box functions. In *Advances in Neural Information Processing Systems*, pp. 918–926.

Horn, D., & Bischl, B. (2016). Multi-objective parameter configuration of machine learning algorithms using model-based optimization. In *2016 IEEE Symposium Series on Computational Intelligence (SSCI)*, pp. 1–8.

Huband, S., Barone, L., While, L., & Hingston, P. (2005). A scalable multi-objective test problem toolkit. In *International Conference on Evolutionary Multi-Criterion Optimization*, pp. 280–295.

Huband, S., Hingston, P., Barone, L., & While, L. (2006). A review of multiobjective test problems and a scalable test problem toolkit. *IEEE Transactions on Evolutionary Computation*, *10*(5), 477–506.

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

Igel, C. (2005). Multi-objective model selection for support vector machines. In *International Conference on Evolutionary Multi-Criterion Optimization*, pp. 534–546.

Jeong, S., Minemura, Y., & Obayashi, S. (2006). Optimization of combustion chamber for diesel engine using Kriging model. *Journal of Fluid Science and Technology*, *1*(2), 138–146.

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

Kandasamy, K., Krishnamurthy, A., Schneider, J., & Póczos, B. (2018). Parallelised Bayesian optimisation via Thompson sampling. In *International Conference on Artificial Intelligence and Statistics*, pp. 133–142. PMLR.

Knowles, J. (2006). ParEGO: A hybrid algorithm with on-line landscape approximation for expensive multiobjective optimization problems. *IEEE Transactions on Evolutionary Computation*, *10*(1), 50–66.

Krizhevsky, A. (2009). Learning multiple layers of features from tiny images. Master's thesis, University of Tront.

Kushner, H. J. (1964). A new method of locating the maximum point of an arbitrary multipeak curve in the presence of noise. *Journal of Basic Engineering*, *86*, 97–106.

Li, L., Jamieson, K., DeSalvo, G., Rostamizadeh, A., & Talwalkar, A. (2017). Hyperband: A novel bandit-based approach to hyperparameter optimization. *Journal of Machine Learning Research*, *18*(1), 6765–6816.

Liu, H., Ong, Y.-S., Shen, X., & Cai, J. (2020). When Gaussian process meets big data: A review of scalable GPs. *IEEE transactions on neural networks and learning systems*, *31*(11), 4405–4423.

López-Ibánez, M., Paquete, L., & Stützle, T. (2010). Exploratory analysis of stochastic local search algorithms in biobjective optimization. In *Experimental Methods for the Analysis of Optimization Algorithms*, pp. 209–222. Springer.

Lu, Z., Whalen, I., Boddeti, V., Dhebar, Y., Deb, K., Goodman, E., & Banzhaf, W. (2019). NSGA-Net: Neural architecture search using multi-objective genetic algorithm. In *Proceedings of the Genetic and Evolutionary Computation Conference*, p. 419–427.

McKay, M. D., Beckman, R. J., & Conover, W. J. (1979). Comparison of three methods for selecting values of input variables in the analysis of output from a computer code. *Technometrics*, *21*(2), 239–245.

Namura, N., Shimoyama, K., & Obayashi, S. (2017). Expected improvement of penalty-based boundary intersection for expensive multiobjective optimization. *IEEE Transactions on Evolutionary Computation*, *21*(6), 898–913.

Nardi, L., Souza, A., Koeplinger, D., & Olukotun, K. (2019). Hypermapper: A practical design space exploration framework. In *2019 IEEE 27th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, pp. 425–426. IEEE.

Nemhauser, G. L., Wolsey, L. A., & Fisher, M. L. (1978). An analysis of approximations for maximizing submodular set functions—I. *Mathematical Programming*, *14*(1), 265–294.

Ozaki, Y., Tanigaki, Y., Watanabe, S., & Onishi, M. (2020). Multiobjective tree-structured Parzen estimator for computationally expensive optimization problems. In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*, pp. 533–541.

Pan, L., He, C., Tian, Y., Wang, H., Zhang, X., & Jin, Y. (2018). A classification-based surrogate-assisted evolutionary algorithm for expensive many-objective optimization. *IEEE Transactions on Evolutionary Computation*, *23*(1), 74–88.

Parsa, M., Mitchell, J. P., Schuman, C. D., Patton, R. M., Potok, T. E., & Roy, K. (2020). Bayesian multi-objective hyperparameter optimization for accurate, fast, and efficient neural network accelerator design. *Frontiers in neuroscience*, *14*, 667.

Parzen, E. (1962). On estimation of a probability density function and mode. *The Annals of Mathematical Statistics*, *33*(3), 1065–1076.

Pelamatti, J., Brevault, L., Balesdent, M., Talbi, E.-G., & Guerin, Y. (2021). Bayesian optimization of variable-size design space problems. *Optimization and Engineering*, *22*(1), 387–447.

Picheny, V. (2015). Multiobjective optimization using Gaussian process emulators via stepwise uncertainty reduction. *Statistics and Computing*, *25*(6), 1265–1280.

Ponweiser, W., Wagner, T., Biermann, D., & Vincze, M. (2008). Multiobjective optimization on a limited budget of evaluations using model-assisted $\mathcal{S}$-metric selection. In *International Conference on Parallel Problem Solving from Nature*, pp. 784–794.

Qian, P. Z. G., Wu, H., & Wu, C. J. (2008). Gaussian process models for computer experiments with qualitative and quantitative factors. *Technometrics*, *50*(3), 383–396.

Rahat, A. A., Everson, R. M., & Fieldsend, J. E. (2017). Alternative infill strategies for expensive multi-objective optimisation. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 873–880.

Rasmussen, C. E., & Williams, C. K. (2006). *Gaussian Processes for Machine Learning*. MIT Press Cambridge, MA.

Sacks, J., Welch, W. J., Mitchell, T. J., & Wynn, H. P. (1989). Design and analysis of computer experiments. *Statistical Science*, *4*(4), 409–435.

Shah, A., & Ghahramani, Z. (2016). Pareto frontier learning with expensive correlated objectives. In *International Conference on Machine Learning*, pp. 1919–1927.

Snoek, J., Larochelle, H., & Adams, R. P. (2012). Practical Bayesian optimization of machine learning algorithms. In *Advances in Neural Information Processing Systems*, pp. 2951–2959.

Wang, Z., & Jegelka, S. (2017). Max-value entropy search for efficient Bayesian optimization. In *International Conference on Machine Learning*, pp. 3627–3635.

Zhan, D., Cheng, Y., & Liu, J. (2017). Expected improvement matrix-based infill criteria for expensive multiobjective optimization. *IEEE Transactions on Evolutionary Computation*, *21*(6), 956–975.

Zhang, Q., & Li, H. (2007). MOEA/D: A multiobjective evolutionary algorithm based on decomposition. *IEEE Transactions on Evolutionary Computation*, *11*(6), 712–731.

Zhang, Q., Liu, W., Tsang, E., & Virginas, B. (2009). Expensive multiobjective optimization by MOEA/D with Gaussian process model. *IEEE Transactions on Evolutionary Computation*, *14*(3), 456–474.

Zhang, Y., Apley, D. W., & Chen, W. (2020). Bayesian optimization for materials design with mixed quantitative and qualitative variables. *Scientific reports*, *10*(1), 1–13.

Zitzler, E., Thiele, L., Laumanns, M., Fonseca, C. M., & Da Fonseca, V. G. (2003). Performance assessment of multiobjective optimizers: An analysis and review. *IEEE Transactions on evolutionary computation*, *7*(2), 117–132.

Zuluaga, M., Krause, A., & Püschel, M. (2016). ε-PAL: an active learning approach to the multi-objective optimization problem. *Journal of Machine Learning Research*, *17*(1), 3619–3650.

Zuluaga, M., Sergent, G., Krause, A., & Püschel, M. (2013). Active learning for multi-objective optimization. In *International Conference on Machine Learning*, pp. 462–470.