Adapting Multi-objectivized Software Configuration Tuning

Chen, Tao; Li, Miqing

DOI:
10.1145/3643751

Document Version
Publisher's PDF, also known as Version of record

Citation for published version (Harvard):
Chen, T & Li, M 2024, 'Adapting Multi-objectivized Software Configuration Tuning', *Proceedings of the ACM on Software Engineering*, vol. 1, no. FSE, 25, pp. 539-561. https://doi.org/10.1145/3643751

Adapting Multi-objectivized Software Configuration Tuning

TAO CHEN[*†], UESTC, China and University of Birmingham, United Kingdom
MIQING LI[‡], University of Birmingham, United Kingdom

When tuning software configuration for better performance (e.g., latency or throughput), an important issue that many optimizers face is the presence of local optimum traps, compounded by a highly rugged configuration landscape and expensive measurements. To mitigate these issues, a recent effort has shifted to focus on the level of optimization model (called meta multi-objectivization or MMO) instead of designing better optimizers as in traditional methods. This is done by using an auxiliary performance objective, together with the target performance objective, to help the search jump out of local optima. While effective, MMO needs a fixed weight to balance the two objectives—a parameter that has been found to be crucial as there is a large deviation of the performance between the best and the other settings. However, given the variety of configurable software systems, the "sweet spot" of the weight can vary dramatically in different cases and it is not possible to find the right setting without time-consuming trial and error. In this paper, we seek to overcome this significant shortcoming of MMO by proposing a weight adaptation method, dubbed AdMMO. Our key idea is to adaptively adjust the weight at the right time during tuning, such that a good proportion of the nondominated configurations can be maintained. Moreover, we design a partial duplicate retention mechanism to handle the issue of too many duplicate configurations without losing the rich information provided by the "good" duplicates.

Experiments on several real-world systems, objectives, and budgets show that, for 71% of the cases, AdMMO is considerably superior to MMO and a wide range of state-of-the-art optimizers while achieving generally better efficiency with the best speedup between 2.2× and 20×.

CCS Concepts: • **Software and its engineering → Search-based software engineering**; **Software performance**; *Software configuration management and version control systems.*

Additional Key Words and Phrases: Configuration tuning, performance optimization, search-based software engineering, multi-objectivization

ACM Reference Format:
Tao Chen and Miqing Li. 2024. Adapting Multi-objectivized Software Configuration Tuning. *Proc. ACM Softw. Eng.* 1, FSE, Article 25 (July 2024), 23 pages. https://doi.org/10.1145/3643751

1 INTRODUCTION

Have you ever struggled in configuring your complex software system? There is no need to worry as we have automatic software configuration tuning. Indeed, it has been shown that 59% of the most severe performance bugs are caused by poor configuration [Han and Yu 2016], making it one of the most dangerous threats to software quality. Over the past decade, software configuration tuning has been an important way to optimize the most important performance concern, such as

[*]This work was conducted when visiting UESTC.
[†]Tao Chen is the corresponding author (t.chen@bham.ac.uk).
[‡]Both authors make commensurable contributions to this research.

Authors' addresses: Tao Chen, UESTC, China and University of Birmingham, Birmingham, United Kingdom, t.chen@bham.ac.uk; Miqing Li, University of Birmingham, Birmingham, United Kingdom, m.li.8@bham.ac.uk.

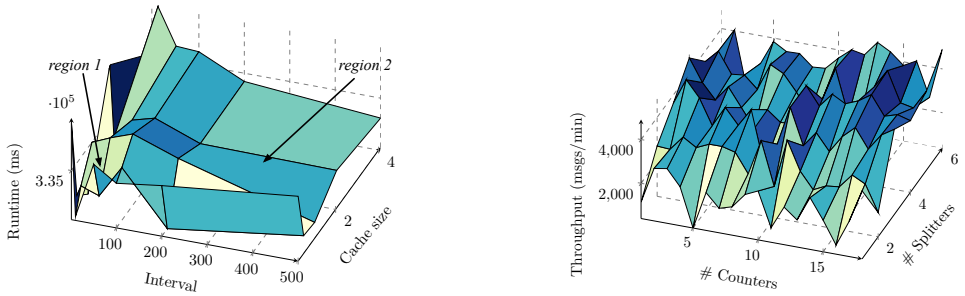
ACM 2994-970X/2024/7-ART25
https://doi.org/10.1145/3643751

runtime, throughput, or accuracy [Behzad et al. 2013; Bergstra et al. 2011; Cáceres et al. 2017; Chen et al. 2021; Chen and Bahsoon 2017b; Chen et al. 2018; Chen and Li 2021; Li et al. 2020a,b; Nair et al. 2020; Shahbazian et al. 2020; Silva-Muñoz et al. 2021; Zuluaga et al. 2016]. However, an unpleasant issue which an optimizer faces is that the tuning may easily be trapped at local optima [Chen et al. 2024; Chen 2022b; Chen and Li 2021; Ding et al. 2015; Li et al. 2014; Zhu et al. 2017]—some configurations that perform better than all the neighbors but being undesirably sub-optimal. While this issue may well be inherited and common in Search-Based Software Engineering (SBSE), there are some unique characteristics in software configuration tuning that make it particularly hard to tackle: (1) the measurement of configurations is expensive. Valov *et al.* [Valov et al. 2017] report that it takes 1,536 hours to explore merely 11 options of configuration for x264. (2) The configuration landscape is rather sparse and rugged, i.e., the close configurations can also have radically different performance [Chen et al. 2024; Chen and Li 2021, 2023b; Gong and Chen 2023; Nair et al. 2020] (see Figure 1). This is because, for example, switching the data structures is merely a single digit change of an option, but that may affect the performance drastically [Chen and Li 2023a; Nair et al. 2020].

From the literature, numerous optimizers have been proposed to tune software configurations. Among others, most of those are mainly guided by direct measurement of the systems, such as Genetic Algorithm [Behzad et al. 2013; Chen et al. 2018; Shahbazian et al. 2020], IRACE [Cáceres et al. 2017; López-Ibánez et al. 2016; Silva-Muñoz et al. 2021], and Random Search [Bergstra and Bengio 2012]. Local optima are handled by search operators, e.g., increasing the neighborhood size [Behzad et al. 2013], or by search strategies that balance exploration and exploitation, such as allowing sampling on less promising regions [Cáceres et al. 2017; López-Ibánez et al. 2016; Silva-Muñoz et al. 2021]. In contrast, surrogate-guided optimizers based on, e.g., Bayesian optimization, also exist [Bergstra et al. 2011; Chen et al. 2021; Nair et al. 2020]. Yet, a major limitation of those optimizers is that they may still easily get stuck at sparse regions of local optima (e.g., if the technique for handling local optima is not particularly effective) or cannot find better configurations efficiently (e.g., if the exploration is overemphasized) [Chen et al. 2024; Chen 2022a; Chen and Li 2021].

Recently, Chen et al. [2024]; Chen and Li [2021] proposed a meta multi-objectivization (MMO) model to tackle the above limitation. Unlike others who work on the design of optimizers, MMO focuses on the optimization model. This is achieved by transforming the tuning into a meta-objective space that involves linear combinations (by a weight) of two performance objectives: one is the target that is of concern and the other is an auxiliary that is of no interest to the software engineers for the considered system. Through constructing two conflicting meta-objectives, MMO keeps a good tendency towards the best target performance objective while preserving high-quality dissimilar configurations, which helps to mitigate local optima.

Despite the promising effectiveness of MMO, Chen et al. [2024]; Chen and Li [2021] also revealed that its performance can be highly sensitive to the weight—a key parameter that impacts the balances between finding better target-objective configuration and diversifying configurations during the tuning. A small weight puts an emphasis on optimizing the target performance objective (exploitation) while a large weight encourages exploration in the search space. It has been shown that an inappropriate weight setting can lead to rather poor results [Chen et al. 2024; Chen and Li 2021]. However, finding a good weight on a given configurable software system is not easy. The "sweet spot" of the weight value, which achieves the ideal balance between exploitation and exploration of the tuning, varies dramatically on different systems or even the status of the tuning, due to the difference in their configuration landscape, scale, and types of options *etc.* For example, Figure 1 shows the different landscapes of two systems. Although both landscapes are rugged, relatively, a smaller weight that favors exploitation more might be preferred to stress finding better performance in Figure 1a since the landscape is smoother; in contrast, in Figure 1b, the weight value might need to be larger so that more exploration can be encouraged to overcome local optima



(a) MongoDB (b) Storm

Fig. 1. The projected configuration landscapes of two configurable software systems.

due to the higher ruggedness. The same applied within one system, e.g., in Figure 1a, when the tuning is trapped at *region 1* then a larger weight is needed to jump out from the area while a smaller weight is better when the tuning is located at *region 2*, as the local landscape is relatively easier to tackle. As such, it is not possible to infer the right weight setting without time-consuming profiling beforehand.

To overcome the above limitation of MMO for software configuration tuning, in this work, we propose a weight adaptation method, dubbed Adaptive MMO or AdMMO. AdMMO does not require any prior knowledge or effort on setting the weight, and the weight is adjusted adaptively as the tuning proceeds. Specifically, our contributions are:

- Through geometric transformation and empirical evidence, we explain how MMO's weight [Chen et al. 2024; Chen and Li 2021] can impact the tuning and why a fixed value is harmful.
- We propose a way to adapt the weight dynamically in MMO on the fly, aiming to maintain an appropriate proportion of the unique nondominated configurations[1] throughout the tuning, which serves as a better indicator of the ideal balance between exploitation and exploration.
- To avoid overfitting the weight, we design a progressive trigger that only enables weight adaptation when needed.
- To mitigate the detrimental effects caused by having too many duplicate configurations[2], we propose a novel partial duplicate retention mechanism that de-emphasizes the duplicate configurations but still allows some good ones to be preserved in the tuning.

Through extensive experiments on 14 system-objective pairs of commonly used real-world software systems and performance objectives [Chen et al. 2024; Chen and Li 2021; Jamshidi et al. 2018; Mendes et al. 2020; Nair et al. 2020; Siegmund et al. 2012] along with different budget sizes, we compare AdMMO with the improved version of original MMO [Chen et al. 2024], the plain multi-objectivization model used in other SBSE problems [Derakhshanfar et al. 2020; Mkaouer et al. 2014; Soltani et al. 2018; Yuan and Banzhaf 2020], seven state-of-the-art optimziers for configuration tuning [Bergstra and Bengio 2012; Cáceres et al. 2017; Chen et al. 2021; Hutter et al. 2011, 2009; Nair et al. 2020; Shahbazian et al. 2020] and three of its variants. Empirically, we have also examined whether there exists a generally best proportion of nondominated configurations to maintain in AdMMO. The results are encouraging, from which we demonstrate that AdMMO:

- is effective as it outperforms state-of-the-art optimizers, including the improved MMO, in 71% of the cases with the best normalized improvements between 17% and 100%;

[1]A configuration $\overline{\mathbf{a}}$ is dominated by $\overline{\mathbf{b}}$ if all objectives of $\overline{\mathbf{b}}$ are better or equivalent to those of $\overline{\mathbf{a}}$ while there is at least one objective of $\overline{\mathbf{b}}$ performs better than that of $\overline{\mathbf{a}}$. A configuration is nondominated if it cannot be dominated by others in the set.
[2]By duplicate configurations, we refer to those with identical values in the configuration space rather than those of the same performance values, since it is still possible for different configurations to have identical performance.

- is efficient, achieving considerable speedup over others with the best from 2.2× to 20×;
- can indeed benefit from the proposed progressive trigger and partial duplicate retention;
- exists a generally best proportion of nondominated configurations to maintain according to this study, i.e., around 30%.

The rest of this paper is organized as follows. Section 2 introduces some background information and related work. Section 3 explains the role of MMO weight, the limitation, and why it is challenging to address it. Section 4 elaborates the designs of our AdMMO. Sections 5 and 6 present our experiment methodology and a detailed analysis of the results, respectively. Section 7 discusses the threats to validity followed by Section 8 that concludes the paper and sheds light on future work.

2 BACKGROUND AND RELATED WORK

In this section, we present the background and discuss the existing methods compared in this work.

2.1 Tuning Software Configuration

A configurable software system often comes with a set of critical configuration options to tune, for example, Storm allows one to change the `num_splitters` and `num_counters` for better latency or throughput [Chen and Li 2021; Nair et al. 2020]. x_i denotes the ith option, which can be either a binary or integer variable, among n options for a software system. The goal we considered in this work is to search for better software configurations, from the space of $\boldsymbol{\mathcal{X}}$, that optimize a single performance objective f[3]:

$$\text{argmin } f(\boldsymbol{x}),\ \boldsymbol{x} \in \boldsymbol{\mathcal{X}} \tag{1}$$

where $\boldsymbol{x} = (x_1, x_2, ..., x_n)$. The measurement of f depends on the target system and the performance attribute, for which we make no assumption about the characteristics in this work.

2.2 Measurement-Based Optimizer

Measurement-based optimizers tune the configuration by solely measuring the software [Chen and Li 2021]. In that regard, many optimizers that leverage different search algorithms exist [Behzad et al. 2013; Chen et al. 2018; Ding et al. 2015; Li et al. 2014; Shahbazian et al. 2020]. Here, we briefly introduce some state-of-the-art optimizers which we compared against in our experiments:

- **Iterated Racing (IRACE)**: IRACE [Cáceres et al. 2017; López-Ibánez et al. 2016; Silva-Muñoz et al. 2021] measures new configurations according to the distributions of configuration options, aiming to jump out of local optima. At each iteration, the distributions are updated using the parent configuration selected and the iteration count, thereby focusing on the search around the best configuration found with more iterations.
- **Genetic Algorithm (GA)**: GA [Behzad et al. 2013; Chen et al. 2018; Shahbazian et al. 2020] is a population-based optimizer that evolves configurations through reproduction. The mutation is used to overcome local optima.
- **Random Search (RS)**: In RS, a configuration is randomly formed and measured during each iteration. Theoretically, it is insensitive to the local optima issue. RS has been shown to be effective in the general configuration domain [Bergstra and Bengio 2012] and it serves as a baseline in this work.
- **ParamILS**: ParamILS [Hutter et al. 2009] iteratively conducts local search around the best configuration found so far but doing so with a probability to jump out from the likely local optima area.

[3]Without loss of generality, we assume minimizing scenarios which can be converted to maximizing via additive inverse.

Nevertheless, the above optimizers may not be effective in finding the best configuration (i.e., global optimum). Some struggle to get rid of local optima and some others fail to find the better configurations efficiently as reported [Chen et al. 2024; Chen and Li 2021].

2.3 Model-Based Optimizer

To guide the tuning, model-based optimizers do not solely rely on the measurements, but also on a gradually updated surrogate [Chen and Bahsoon 2017a; Gong and Chen 2022, 2024] that can cheaply predict the performance [Bergstra et al. 2011; Chen et al. 2021; Jamshidi and Casale 2016; Nair et al. 2020; Zuluaga et al. 2016]. Typically, they follow the procedure of Bayesian Optimization or its variants. In this work, we experimentally examine three model-based optimizers, namely:

- **Flash**: Published at TSE [Nair et al. 2020], Flash extends the Bayesian Optimization by using a regression tree as the surrogate and conducting acquisition without uncertainty.
- **BOCA**: Another optimizer based on Bayesian Optimization from ICSE [Chen et al. 2021]. BOCA uses Random Forest as the surrogate with Expected Improvement. Further, it prioritizes sampling of the important configuration options based on Gini importance from the surrogate.
- **SMAC**: As a general purpose optimizer published at LION [Hutter et al. 2011], SMAC improves ParamILS with a Random Forest model, hence making it a model-based optimizer.

We selected the above because (1) they are state-of-the-art optimizers from both the software engineering and general optimization community [Bartz-Beielstein et al. 2021; Chen et al. 2021, 2024; Chen and Li 2021]. (2) It has been reported that, for software configuration tuning, they outperform other optimizers, e.g., BOCA is better than TPE [Chen et al. 2021] and Flash is superior to ϵ-PAL [Nair et al. 2020]. However, the model-based optimizers suffer from local optima, and prior study has shown that the inaccuracy of the surrogate can cause serious issue [Zhu et al. 2017].

2.4 Plain Multi-Objectivization (PMO)

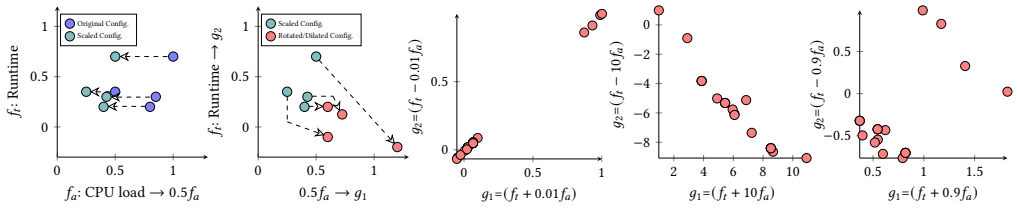
As an alternative way to address the local optimum issue in software configuration tuning for a single performance objective, multi-objectivization assumes two performance objectives to be considered [Chen et al. 2024; Chen and Li 2021]: a target performance objective f_t that is of concern (e.g., runtime) and another auxiliary performance objective f_a that is generally of no interest to the software engineer (e.g., CPU load) on the given system (at least on the occasion of the corresponding tuning round). In other SBSE problems [Derakhshanfar et al. 2020; Mkaouer et al. 2014; Soltani et al. 2018; Yuan and Banzhaf 2020], a natural way of multi-objectivization would be to optimize both f_t and f_a simultaneously, hence leveraging the incomparability of Pareto dominance relation to jump out of the local optima. This, denoted as plain multi-objectivization (PMO), can be expressed as:

$$\text{minimize} \begin{cases} f_a(\boldsymbol{x}) \\ f_t(\boldsymbol{x}) \end{cases} \tag{2}$$

Since PMO is an optimization model, it is agnostic to the underlying multi-objective optimizer. However, a major issue with PMO is that, a configuration with a poor f_t but a good f_a will also be regarded as a good configuration, despite the fact that f_a is of no interest to the software engineer. This does not fit our purpose and can lead to undesired consumption of the tuning budget (due to the unnecessary efforts of optimizing f_a), especially when the measurements are expensive.

2.5 Meta Multi-Objectivization (MMO)

Chen et al. [2024]; Chen and Li [2021] propose a new way of tuning software configuration, dubbed MMO, firstly appeared at FSE'21 [Chen and Li 2021]. Like PMO, MMO is an optimization model and it can also be paired with any multi-objective optimizers, although Chen and Li used NSGA-II [Deb



(a) scaling (b) rotation/dilation (c) $w = 0.01$ (d) $w = 10$ (e) $w = 0.9$

Fig. 2. The theoretical analysis and empirical evidence on the role of MMO weights for system MariaDB with normalized target and auxiliary performance objectives (f_t is *runtime* and f_a is *CPU load*). (a) shows the scaling on f_a by a factor of $w = 0.5$; (b) is the 45° rotation and dilation on both axes by a factor of $\sqrt{2}$ afterwards. (c), (d), and (e) are the empirical results on configurations in an iteration with distinct w thereof.

et al. 2002]—an extremely common multi-objective optimizer in SBSE—as the default. Yet, it does not treat f_t and f_a equally. That is, MMO optimizes f_t whilst diversifying the values of f_a during the tuning via the following model[4]:

$$\text{minimize} \begin{cases} g_1(\boldsymbol{x}) = f_t(\boldsymbol{x}) + wf_a(\boldsymbol{x}) \\ g_2(\boldsymbol{x}) = f_t(\boldsymbol{x}) - wf_a(\boldsymbol{x}) \end{cases} \tag{3}$$

whereby $g_1(\boldsymbol{x})$ and $g_2(\boldsymbol{x})$ are the meta-objectives, each of which shares the same f_t, but differs (effectively being opposite) on f_a. w is a weight parameter that controls the relative contribution. As such, MMO transforms the tuning from searching in the original objectives into the space of meta-objectives, namely the MMO space. Since the performance objectives may come with radically different scales, both $f_t(\boldsymbol{x})$ and $f_a(\boldsymbol{x})$ need to be normalized.

Essentially, MMO aims to seek a diverse set of configurations with two characteristics: (1) the configurations all have fairly high performance on f_t, and (2) the configurations are of clear dissimilarity on f_a. The first characteristic is as the result of both meta-objectives in Eq. (3) optimizing f_t [Chen and Li 2021], which differs from the PMO, where both f_t and f_a are optimized together. The second characteristic is the result of the two meta-objectives having opposite terms on f_a, making configurations incomparable (i.e., nondominated) [Chen and Li 2021]—the key to mitigate local optima in the highly rugged and sparse configuration landscape, which allows MMO to outperform other optimizers [Chen et al. 2024; Chen and Li 2021].

However, Chen et al. [2024]; Chen and Li [2021] have also revealed that in MMO the weight w is a highly sensitive parameter to the performance. Even with the improved MMO[5] under a new improved normalization [Chen et al. 2024], the resolution is simply fixing $w = 1$ as opposed to adapt it (which is our aim).

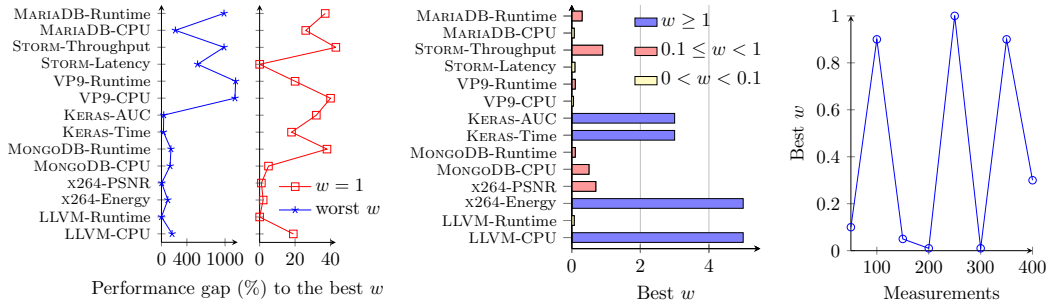
3 WHAT IS WRONG WITH MMO?

Here, we justify what role the weight plays in MMO and why its adaptation in tuning is beneficial.

3.1 Theory: Interpreting the Weight in MMO

Compared with PMO, MMO essentially does two main geometric operations to transform the original space of two performance objectives into a meta-objective space: (1) it scales (stretches or shrinks) the configurations along the f_a axis by a factor of w (Figure 2a); (2) it rotates the scaled

[4]We use the simplest linear form of MMO, as Chen and Li [2021] showed that different forms perform similarly.
[5]Throughout the paper, we use MMO refers to both the original [Chen and Li 2021] and improved MMO [Chen et al. 2024], as they only differ on whether fixing $w = 1$ and the normalization. Yet, we apply the improved MMO for all experiments and as the basis of AdMMO.



(a) gap to best w (b) best w per system-objective pair (c) best w per budget

Fig. 3. The sensitivity of (improved) MMO to weight w over 50 runs on different system-objective pairs. (a) shows the % improvement that can be made by the best w. (b) illustrates the best fixed w. (c) demonstrates the best w throughout the tuning for system MariaDB using *runtime* as the target. The best w (among $\{0.01, 0.03, ..., 10\}$) is identified by Scott-Knott test [Scott and Knott 1974] and the average performance results.

configurations by 45° clockwise and then dilates them on both f_t and f_a by a factor of $\sqrt{2}$ (Figure 2b). Geometrically, MMO can be decomposed via the following transformation metrics in linear algebra:

$$\begin{bmatrix} g_1(\boldsymbol{x}) \\ g_2(\boldsymbol{x}) \end{bmatrix} = \overbrace{\sqrt{2}\begin{bmatrix} \cos\frac{\pi}{4} & \sin\frac{\pi}{4} \\ -\sin\frac{\pi}{4} & \cos\frac{\pi}{4} \end{bmatrix}}^{\text{rotation/dilation matrix}} \overbrace{\begin{bmatrix} w & 0 \\ 0 & 1 \end{bmatrix}}^{\text{scaling matrix}} \overbrace{\begin{bmatrix} f_a(\boldsymbol{x}) \\ f_t(\boldsymbol{x}) \end{bmatrix}}^{\text{original space}} = \overbrace{\begin{bmatrix} f_t(\boldsymbol{x}) + wf_a(\boldsymbol{x}) \\ f_t(\boldsymbol{x}) - wf_a(\boldsymbol{x}) \end{bmatrix}}^{\text{MMO space}} \tag{4}$$

The scaling based on weight w in MMO, together with the rotation/dilation, impacts the balance between enhancing f_t and diversifying configurations via f_a during tuning. Since w can be arbitrarily predefined, its value plays a more critical role therein. A smaller w means a bigger shrink on f_a, thus more configurations become comparable after the rotation/dilation, putting more emphasis on optimizing f_t, i.e., ***exploitation***. In the extreme case when $w = 0$, f_a is completely ruled out, leaving the two meta-objectives identical, and as such all configurations are comparable provided that they differ on f_t. On the other hand, a larger w suggests a bigger stretch on f_a, making more of the configurations become incomparable following the rotation/dilation, which encourages the ***exploration*** in the search space to find more diverse configurations. In the extreme case where $w = \infty$, the differences between configurations on f_a become infinitely large, rendering the two meta-objectives linearly conflicted, hence all configurations are incomparable if they differ on f_a.

Figure 2c to 2e demonstrate that distinct weight values can lead to different states of the configurations during the tuning even with the improved MMO [Chen et al. 2024]. When $w = 0.01$ (Figure 2c), all configurations are comparable on the two meta-objectives. In this case, the problem degenerates to a single-objective problem, thus the search may easily be trapped in local optima. On the other extreme, when $w = 10$ (Figure 2d), almost all configurations are incomparable (i.e. nondominated) in terms of the two meta-objectives, meaning that there is no selection pressure (discriminative power) driving the search toward the Pareto front. In contrast, when $w = 0.9$ (Figure 2e), configurations are of more variety in terms of their quality—they can be superior, inferior, or incomparable between superior/inferior ones, which tends to favor exploitation while still maintaining a certain level of exploration. This also suggests that an ideal proportion of nondominated configurations also implies an appropriate balance between exploitation and exploration

3.2 Limitation of the Fixed Weight in MMO and Why It Is Challenging

The above analysis explains what makes the weight in MMO [Chen et al. 2024; Chen and Li 2021] important: its value determines the proportion of nondominated configurations in the tuning, which reflects the relationship between exploitation and exploration in the tuning hence can severely influence the performance of MMO. In subsequent work, Chen et al. [2024] have acknowledged such and attempted to fix the issue by removing the weight from MMO with an improved normalization scheme, i.e., setting $w = 1$. However, there is still no guarantee that $w = 1$ can help to achieve the reasonable proportion of nondominated configuration for all systems/performance objectives, hence blurring the full potential of MMO. Indeed, from Figure 3a (we use the improved MMO with the new normalization from [Chen et al. 2024]), we see that the performance gap between the worst and best w can be rather high—up to 1167%. Even when setting $w = 1$, the truly best setting can still lead to considerable improvement in general with up to 43% (throughput on Storm).

Resolving the weight setting issue is not easy, because for different systems/performance objectives, the "sweet spot" (best setting) of the weight, which achieves a reasonably balanced proportion of nondominated configurations during the tuning, can be very disparate due to the difference of configuration landscape and scale, *etc* [Chen et al. 2024; Chen and Li 2021]. From Figure 3b, we note that the optimal weight value varies dramatically, e.g., the best w value for the *runtime* of MongoDB and the *CPU load* of LLVM are 0.05 and 5, respectively—a 100× difference. This brings a big difficulty to the attempt to set the right weight via trial and error, which itself is time-consuming due to the expensive measurement of configurations.

On top of that, even for the same system/performance objective, different budgets/stages of the tuning can have distinct optimal weight settings. When the configurations concentrate in a small region of the search space, a large weight can be helpful to jump out of the local optimum. When the configurations scatter widely over the space, a small weight can be beneficial to steer the tuning towards the right direction (i.e., better target performance). Figure 3c gives the optimal weight settings during different tuning budgets/stages (every 50 measurements) for the system MariaDB. As can be seen, when the tuning consumes 200 measurements, a tiny weight value is the best, whereas after 50 measurements at a budget of 250, the weight being 1 is the best.

The above suggests that a fixed weight setting for all cases and throughout the tuning is not ideal, hence an adaptive weight, which can change in line with the tuning, is needed for dynamically maintaining a good proportion of nondominated configurations—a neither too high nor too low weight value that indicates a good balance of exploitation and exploration, e.g., in Figure 2e.

4 SOFTWARE CONFIGURATION TUNING WITH ADAPTIVE MMO

We now delineate the designs of AdMMO. As in Algorithm 1, similar to MMO, AdMMO also uses NSGA-II [Deb et al. 2002] as the underlying multi-objective optimizer and the tuning runs in the transformed MMO space. The extensions of AdMMO to the original/improved MMO are three-fold:

- We maintain a good proportion of unique nondominated configurations by dynamically adapting the weight in MMO (line 22);
- but doing so only when it is necessary (line 21);
- while preserving the promising configurations with partially retained duplicates (line 25).

In what follows, we elaborate on these designs in detail.

4.1 Progressive Trigger for Adapting w in MMO during Tuning

Constantly adapting the weight during the tuning is unnecessary, or can even be harmful, as it takes a few consecutive iterations to gather the tuning state. In AdMMO, we design a *progressive*

Algorithm 1: Pseudo-code of AdMMO (with NSGA-II [Deb et al. 2002] as the base optimizer)

Input: Configuration space $\mathcal{V}$; system $\mathcal{S}$; initial weight in MMO $w = 1$; budget B; offset $T = 1$; cut-off point $C = 0.5$; expected proportion of nondominated configuration p
Output: the best configuration on f_t

1 Randomly initialize a population of n configurations $\mathcal{P}$
2 measure($\mathcal{P}, \mathcal{S}$) /* measuring f_t and f_a. */
3 normalize($\mathcal{P}$)
4 $\mathcal{U} \leftarrow$computeMMO($\mathcal{P}, w$) /* computing MMO meta-objectives g_1 and g_2. */
5 $b = b + n$
6 **while** $b < B$ **do**
7 $\mathcal{P}' = \emptyset$
8 **while** $\mathcal{P}' < n$ **do**
9 $\{p_x, p_y\} \leftarrow$mating($\mathcal{P}$) /* mating based on g_1 and g_2. */
10 $\{s_x, s_y\} \leftarrow$doCrossoverAndMutation($\mathcal{V}, p_x, p_y$)
11 **for** $\forall s_i \in \{s_x, s_y\}$ *that is new* **do**
12 measure($s_i, \mathcal{S}$) /* new configurations are measured; use previous measurement otherwise. */
13 $b = b + 1$
14 $\mathcal{P}' = \mathcal{P}' \bigcup \{s_x, s_y\}$
15 **if** *a new best configuration on* f_t *found in* $\mathcal{P}'$ **then**
16 o = 0
17 **else**
18 o = o + 1
19 $\mathcal{U} = \mathcal{P} \bigcup \mathcal{P}'$
20 normalize($\mathcal{U}$) /* the new local normalization as in the improved MMO [Chen et al. 2024] */
21 **if** isTrigger(o, T, b, B, C) **then** /* triggering adaptation via Equation 5. */
22 $\mathcal{U} \leftarrow$runMMOwithAdaptiveWeight($\mathcal{U}, w, p$) /* adapting w to compute g_1 and g_2. */
23 **else**
24 $\mathcal{U} \leftarrow$runMMO($\mathcal{U}, w$) /* computing g_1 and g_2 with current w. */
25 $\mathcal{P} \leftarrow$MMOwithPartialDuplicates($\mathcal{U}$)/* survival selection with partial duplicate retention. */
26 **return** the configuration with the best f_t in $\mathcal{P}$

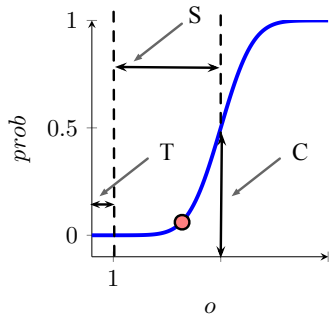
trigger to determine when a weight adaptation is needed (at line 21 in Algorithm 1). In general, our trigger design is derived from two observations from tuning software configuration:

- The less the measurements (i.e., at an earlier stage of tuning), the less likely that the weight adaptation is needed as there is a smaller chance that the tuning has stuck (either due to local optima or limited discriminative power).
- If a better configuration can still be discovered, then it is less desirable to adapt the weight since this implies that the current proportion of nondominated configurations can still be effective in guiding the tuning.

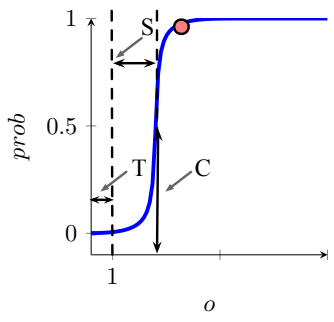
As a result, at each iteration of the tuning, we compute the probability of triggering weight adaptation, *prob*, using the following function (with an illustration in Figure 4):

$$prob = 1 - \exp\left(\frac{\ln C \times \max(0, o - T)}{S^2}\right) \tag{5}$$

whereby o is the number of consecutive iterations that no better configuration is found and T is a given offset, indicating the tolerance level of o (which we set the most restricted value of $T = 1$ in this work). Hence the more iterations that the tuning gets stuck in a configuration, the more likely

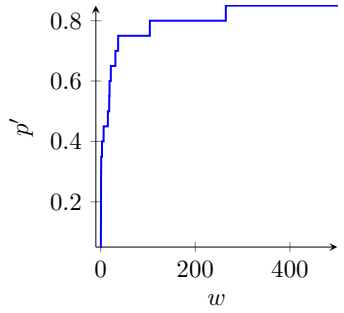


(a) $T = 1, C = 0.5, S = 8$ ($B = 400$; $b = 50$)

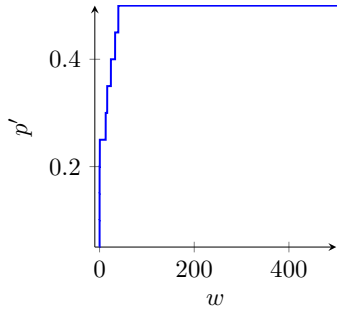


(b) $T = 1, C = 0.5, S = 4$ ($B = 400$; $b = 100$)

Fig. 4. The probability distribution in progressive trigger of AdMMO. The red dot denotes $o = 5$: 5 consecutive iterations in which the best configuration has not been changed. Clearly, even with the same o, there is a higher probability of trigger in (b) than (a) as the former consumes more measurements.



(a) Storm



(b) MongoDB

Fig. 5. Correlation between the actual proportion of nondominated configurations p' and the weight w under the (improved) MMO in one run (population size of 10 and without duplicates). f_t and f_a of Storm are throughput and latency, receptively; for MongoDB, the f_t is runtime while f_a is CPU load.

that we need to adapt the weight. S determines the slope of the function, which is calculated by the ratio between the budget (B) and the current number of measurements (b): $S = \frac{B}{b}$. As such, S decreases as more configurations are measured, meaning that the steeper the slope and hence the more likely it is to trigger weight adaptation for the same o value. C is a cut-off probability that also influences the slope of the shape and we use the most pragmatic setting of 0.5 (50% probability). In this way, we progressively increase the probability of adapting the weight proportional to the number of measurements (by S) and the iteration count with no better configuration found (by o).

4.2 Maintaining Nondominated Configurations in the MMO Space

As discussed in Section 3, the proportion of nondominated configurations is controllable via the weights in MMO, hence under each trigger, we seek to adapt the weights such that the proportion, denoted as p', reaches a given level p (line 22 in Algorithm 1). To this end, our design was derived from the theoretical analysis in Section 3 and some important properties for tuning software configuration in that regard, as illustrated by the exampled systems from Figure 5:

Algorithm 2: runMMOwithAdaptiveWeight

Input: Union of current population and the generated offsprings $\mathcal{U}$; current weight w; expecting proportion of nondominated configurations p
Output: Configurations with MMO meta-objectives under adapted w
Declare: $\Delta = 0.1$; bounds of weight: $w_{max} = 10^3$; $w_{min} = 0$

```
1  while p' ≠ p and w_min < w < w_max do
2      computeMMO(U, w)
3      U' ←removeDuplicates(U)
4      F ←nondominatedSorting(U')
5      p' = getProportionOfNondominatedConfigurations(F)
6      if p' < p then
7          if w + Δ ≥ 0.1 then
8              Δ = 0.1
9          w = w + Δ
10     else if p' > p then
11         if w − Δ < 0.1 then
12             Δ = 0.1 × 10^−3
13         w = w − Δ
14 return U
```

- Clearly, when the weight increases, the proportion would never become smaller, and similarly, decrements of the weight would never make the proportion larger (recall Section 3.1).
- Although changing MMO weight can control the proportion of nondominated configurations, the sensitivity of which can differ depending on the given system.

Bearing the above in mind, AdMMO adapts the weight via the following steps (Algorithm 2):

(1) Examine the current proportion of unique nondominated configurations[6] $p' = \frac{n_d}{n_u}$ under w by using the count of unique nondominated configurations (n_d) together with the size of the current unique population and offsprings n_u (lines 2-5 and see Section 4.3).
(2) If $p' < p$, we increase w by Δ; if $p' > p$, we decrease w by Δ. The Δ is updated depending on two situations, which we found appropriate (lines 6-9 and 10-13):

$$\Delta = \begin{cases} 0.1, \text{ if } p' < p \text{ and } w + \Delta \geq 0.1 \\ 0.1 \times 10^{-3}, \text{ if } p' > p \text{ and } w - \Delta < 0.1 \end{cases} \tag{6}$$

When $p' = p$, the adaptation terminates.
(3) Repeat from step 1 with the updated w. To prevent adapting forever, we set bounds as $w_{max} = 10^3$ and $w_{min} = 0$ to force the adaptation to stop once either of them is reached, as we found that they produce the most stable results on all systems/performance objectives.

As we will show in Section 6.4, empirically we are able to conclude that $p = 0.3$ (30% proportion) is generally the best and most stable level to maintain across the cases.

4.3 Partial Duplicate Retention for Preserving Promising Configurations

Since MMO preserves the tendency toward the best target performance objective and there is often a high sparsity for configurable software systems [Chen and Li 2021; Gong and Chen 2023; Nair et al. 2020], MMO will likely to accumulate many duplicate configurations. Yet, having many duplicate

[6]The divisor is less important in the calculation as it can be with or without duplicate configurations; we use the one without duplicates for its interpretability. The most important factor is the count of nondominated configurations.

Algorithm 3: MMOwithPartialDuplicates

Input: Union of current population and the generated offsprings $\mathcal{U}$
Declare: The set of duplicate configurations D
Output: New population of configurations $\mathcal{P}$

1 $\mathcal{F}$ = nondominatedSorting($\mathcal{U}$)
2 **for** $\forall F_i \in \mathcal{F}$ *when* $\mathcal{P}$ *is not full* **do**
3 **if** F_{i+1} *exists* **then**
4 D = getDuplicate(F_i)
5 F_i = removeDuplicateConfigurations(F_i,D)
6 $F_{i+1} = F_{i+1} \bigcup D$
7 **if** *the remaining size of* $\mathcal{P} \geq$ *the size of* F_i **then**
8 $\mathcal{P} = \mathcal{P} \bigcup F_i$
9 **else**
10 F'_i = sortByCrowdingDistance(F_i)
11 $\mathcal{P} = \mathcal{P} \bigcup$ top k configurations from F'_i until $\mathcal{P}$ is full
12 **return** $\mathcal{P}$

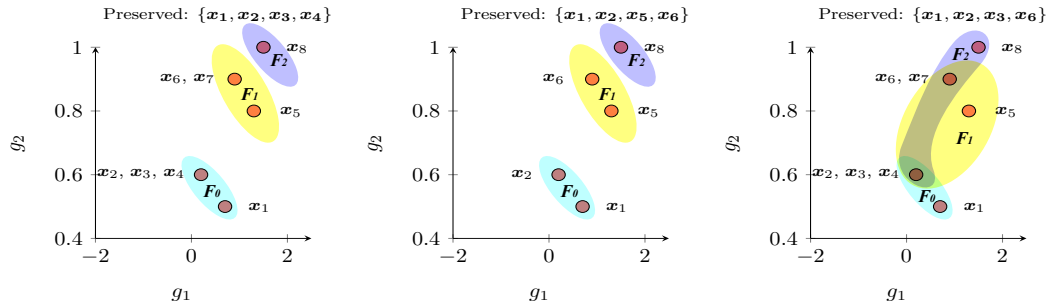
configurations can harm the weight adaptation as the duplicates might also be nondominated. As such, the proportion of nondominated configurations may appear to be appropriate but in fact, most (if not all) of those nondominated ones can be duplicates. Hence, when examining the actual proportion of nondominated configurations p', we need to consider only the unique configurations (line 3 in Algorithm 2).

Next, in the survival selection—a procedure that preserves promising configurations for the next iteration—of the optimizer that underpins AdMMO (i.e., NSGA-II [Deb et al. 2002]), we also need to distinguish the duplicates or otherwise the selection will lose the guidance since there could be too many nondominated yet duplicate configurations. However, unlike the calculation of p', what makes it more challenging is that simply removing all but one duplicate (which is a common method [Fortin and Parizeau 2013]) can also be harmful, as poor configurations dominated by some removed (and better) duplicates may be preserved, thereby undermining the tuning performance. Indeed, our experiments suggest that considering only the unique configurations in the selection is detrimental (see Section 6.3).

To overcome the above, we design a partial duplicate retention mechanism in the survival selection of AdMMO, ensuring that the duplicates are distributed amongst different fronts and the better ones will be with higher-ranked fronts (line 25 in Algorithm 1). As such, high-quality duplicates (i.e., the good ones) still have a chance to survive when there are fewer unique nondominated configurations than the population size (but not always). As shown in Algorithm 3, we perform the steps below to determine which configurations can be preserved to the next iteration:

(1) Run nondominated sort in NSGA-II to produce a vector of fronts $\mathcal{F} = \{F_0, F_1, ..., F_n\}$ (F_0 is the nondominated front) where duplicate configurations will be in the same front (line 1).
(2) For a front F_i where $0 \leq i \leq n-1$, we preserve only the unique configurations in F_i and move all the remaining duplicates D to F_{i+1} (lines 3-6).
(3) Preserve configurations to survive from F_i (lines 7-8). If not all configurations in F_i can be fitted in the next population, select them based on crowding distance (lines 9-11). This is the same as in the original NSGA-II.
(4) Return the population if it is full. Otherwise, repeat from step 2 until the population is full.

Figure 6 gives an example. Suppose that the population size is 4 and after the normal nondominated sorting, we have three fronts: $F_0 = \{x_1, x_2, x_3, x_4\}$, $F_1 = \{x_5, x_6, x_7\}$, and $F_2 = \{x_8\}$,



(a) without distinguish duplicates (b) remove extra duplicates (c) partial duplicate retention

Fig. 6. The results from different ways of handling duplicates when preserving configurations for the next iteration under the MMO meta-objectives space. x_2, x_3, and x_4 are duplicate; x_6 and x_7 are also duplicate.

which contains a rather high number of nondominated yet duplicate configurations. Clearly, when computing p' without duplicates, the number of nondominated configurations considered is 2.

In this case, if we do not distinguish the duplicate nondominated configurations in the selection (Figure 6a), then there would be 4 nondominated configurations, which is inconsistent with the calculation of p', and the preserved ones are $\{x_1, x_2, x_3, x_4\}$ with three duplicates. If we simply remove all duplicates in the selection (Figure 6b), then indeed we can have 2 nondominated configurations, but the preserved ones would be $\{x_1, x_2, x_5, x_6\}$, which is not ideal since both x_5 and x_6 are configurations that have been dominated by others. With partial duplicate retention (Figure 6c), we can have $F_0 = \{x_1, x_2\}$ and it would certainly be preserved. Now, we move x_3 and x_4 to F_1 which would become $F_1 = \{x_5, x_6, x_7, x_3, x_4\}$. After step 2 at this front, we would have $F_1 = \{x_5, x_6, x_3\}$, leaving x_4 and x_7 again to F_2 and having $F_2 = \{x_4, x_7, x_8\}$. Since we only need two more configurations, we stick with F_1 to preserve the top 2 most diverse ones from $\{x_5, x_6, x_3\}$, which will be x_3 and x_6 according to the crowding distance. As such, there would still be 2 nondominated configurations and it is consistent with the calculation of p', while good configurations (x_2) and their duplicates (x_3 and x_4) are distributed across different fronts, some of which are higher-ranked. Here, the preserved ones will be $\{x_1, x_2, x_3, x_6\}$, which is more ideal than the outcome of ignoring the duplicates in terms of diversity since only x_2 and x_3 are duplicate, while also clearly better than the result of simply removing all duplicates as x_3 dominates x_5.

Notably, the above retention mechanism is tailored for AdMMO to use an underlying multi-objective optimizer that is compatible with nondominated sorting, therefore it will directly work with those in the same family of NSGA-II, e.g., SMS-EMOA [Beume et al. 2007] and GrEA [Yang et al. 2013] (the diversity distance therein is flexibly changeable). Although the idea/purpose behind retention is generic, adopting the mechanism (and AdMMO) to pair other families of optimizers, e.g., IBEA [Zitzler and Künzli 2004], requires some amendments, which we leave for future work.

5 EVALUATION

In this section, we elaborate on the experiment settings. All experiments were run on a dedicated High-Performance Computing server with 64-core Intel Xeon 2.6GHz and 256GB DDR RAM.

5.1 Research Questions

Our experimental evaluation seeks to answer four research questions (RQs):

- **RQ1:** How does AdMMO perform against the state-of-the-art optimizers for tuning?
- **RQ2:** How efficient is the AdMMO in utilizing resources over the state-of-the-art optimizers?
- **RQ3:** How beneficial are the progressive trigger and partial duplicate retention?

Table 1. Configurable software systems and performance objectives studied. $|\mathcal{O}|$ and $|\mathcal{S}|$ denote the number of options and search space, respectively.

System	Language	Domain	The Two Performance Objectives	$\lvert\mathcal{O}\rvert$	$\lvert\mathcal{S}\rvert$
MariaDB	C/C++/Perl	SQL database	Runtime and CPU load	10	864
Storm	Java/Clojure	Stream process	Throughput and Latency	6	2,880
VP9	C	Video encoding	Runtime and CPU load	12	3,008
Keras	Python	Deep learning	AUC and Inference time	13	12,288
MongoDB	C++	Non-SQL database	Runtime and CPU load	15	6,840
x264	C	Video encoding	PSNR and Energy consumption	17	53,662
LLVM	C++	Compiler	Runtime and CPU load	16	65,436

- **RQ4:** What is the sensitivity of AdMMO to the maintained proportion of nondominated configurations?

RQ1 helps us to understand the ability of AdMMO to tune the configurations. However, doing so by consuming a large amount of resources (the number of measurements) is clearly undesirable. Therefore, we ask **RQ2** to assess whether AdMMO is efficient in finding promising configurations. **RQ3** allows us to conduct ablation studies of AdMMO, hence the usefulness of our adaptation designs can be examined. We use **RQ4** to study the sensitivity of AdMMO to p.

5.2 Configurable Software Systems and State-of-the-Art Optimizers

In this work, we consider the highly configurable software systems that have been widely used in existing work [Chen et al. 2024; Jamshidi et al. 2018; Mendes et al. 2020; Nair et al. 2020; Siegmund et al. 2012], including the same sets of performance objectives, options and possible configurations. Since evaluating all systems is unrealistic, we select them according to three criteria:

- The systems have been measured on at least two performance objectives.
- The systems have both binary and categorical/numeric options or more than 10 options.
- The systems should have clear instructions on how they are deployed.
- For systems with multiple benchmarks, we use the one with the most deviated performance.

As shown in Table 1, the systems studied are of a diverse nature. To examine AdMMO, we use each of the two performance objectives of a system as the target performance objective in turn while the other serves as the auxiliary performance objective according to Chen and Li [Chen et al. 2024; Chen and Li 2021]. This gives us 14 system-objective pairs in total. All systems are tuned using the benchmarks setting from prior work (e.g., WordCount for Storm) [Chen et al. 2024; Chen and Li 2021; Jamshidi et al. 2018; Mendes et al. 2020; Nair et al. 2020; Siegmund et al. 2012].

We compare all optimizers in Section 2, including the improved MMO [Chen et al. 2024].

5.3 Tuning Budget

To avoid noises from the implementation tricks and the nature of different programming languages underpinning the compared optimizers, we measure the "speed" of tuning by the number of measurements (a language-independent feature) as suggested in prior work [Nair et al. 2020]. Notably, only the measurements of distinct configurations can consume the budgets.

Since the measurement is often expensive, e.g., it can take up to 166 minutes to measure one configuration on MariaDB, the possible budgets are often not unrealistically high compared with some other domains and SBSE problems [Behzad et al. 2013; Chen et al. 2024; Chen and Li 2021; Shahbazian et al. 2020]. To match with the realistic settings, we examine different budget sizes including 100, 200, 300, and 400 measurements, denoted as S_{100}, S_{200}, S_{300}, and S_{400}, respectively. Note that even with S_{100}, in reality, systems like MongoDB will still require more than two full weeks to complete all the unique measurements. As such, the budgets serve as balanced choices between the convergence of optimizers and the limits of available resources in the practical scenarios.

Each measurement is run three times and the average is used. Like prior work [Chen and Li 2021], we stored the measurements as datasets for reuse (and for expediting the experiments).

5.4 Evaluation Metrics

It is worth noting that, since we are interested in optimizing a single performance attribute that is of the most concern, the key evaluation metric would be the achieved performance when the tuning terminates (see Table 1) rather than any multi-objective quality metrics [Li et al. 2022]. We also measure the efficiency of the optimizers in terms of the resources required to achieve a promising performance result, the details of which can be found in Section 6.2.

5.5 Optimizer Settings

Since AdMMO leverages NSGA-II [Deb et al. 2002]—a common multi-objective measurement-based optimizer from SBSE, we apply the binary tournament for mating selection and a population size of 10, together with the boundary mutation and uniformed crossover under the rates of 0.1 and 0.9, respectively, as prior work [Chen et al. 2018, 2019; Shahbazian et al. 2020]. Those settings are often recommended [Chen et al. 2021] and we seek to relieve the dependency on specific components of these optimizers. We set $p = 0.3$ (unless otherwise stated), meaning that AdMMO seeks to maintain 30% of the unique nondominated configurations as this is a generally best setting (we will show in Section 6.4); other (less important) parameters are fixed as stated in Section 4. AdMMO is implemented using jMetal [Durillo and Nebro 2011] and Opt4J [Lukasiewycz et al. 2011].

For GA, MMO (in the experiments, we compare with the latest improved MMO [Chen et al. 2024] that uses $w = 1$ and the improved normalization scheme, as it performs better than the original version [Chen and Li 2021]), and PMO, we set the same settings as those for AdMMO while both MMO and PMO are also paired with NSGA-II. For other optimizers, we use exactly the same setting for IRACE as reported by López-Ibánez et al. [2016] and those for ParamILS from the work of Hutter et al. [2009]. While Flash has no other parameters except for the budgets, BOCA and SMAC require a few settings, for which we use the identical values as those reported in their work [Chen et al. 2021; Nair et al. 2020]. A sampling budget of 30 is used for initializing the model in model-based optimizers as recommended by Nair et al. [2020].

To mitigate stochastic bias, all experiments are repeated 50 runs.

5.6 Statistical Validation

We use the recommended non-parametric Wilcoxon test [Wilcoxon 1945] with $a = 0.05$ to verify the significance of pairwise comparisons between AdMMO and its counterpart over the 50 runs [Arcuri and Briand 2011]. Further, we use $\hat{\mathbf{A}}_{12}$ [Vargha and Delaney 2000] to examine the effect size. According to Vargha and Delaney [Vargha and Delaney 2000], $0.56 \leq \hat{\mathbf{A}}_{12} < 0.64$ (or $0.36 < \hat{\mathbf{A}}_{12} \leq 0.44$) indicates a small yet non-trivial effect size while $0.64 \leq \hat{\mathbf{A}}_{12} < 0.71$ (or $0.29 < \hat{\mathbf{A}}_{12} \leq 0.36$) and $\hat{\mathbf{A}}_{12} \geq 0.71$ (or $\hat{\mathbf{A}}_{12} \leq 0.29$) mean a medium and a large effect size, respectively.

In this work, we say the difference is statistically significant only when $\hat{\mathbf{A}}_{12} \geq 0.56$ (or $\hat{\mathbf{A}}_{12} \leq 0.44$) and $p\text{-}value < 0.05$; otherwise the deviation in the comparison is trivial.

6 RESULTS AND ANALYSIS

We now present and discuss the experiment results.

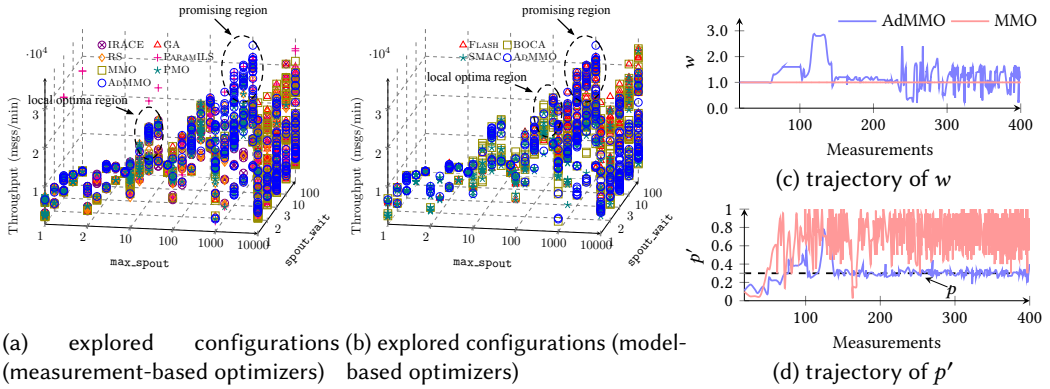
6.1 Effectiveness

6.1.1 Method. For **RQ1**, we compare AdMMO against all optimizers specified in Section 2. Since there are 14 system-objective pairs and 4 different budgets, we obtain 56 cases of comparisons.

Table 2. Comparing the optimizers on the average of normalized target performance over 50 runs (smaller value is preferred). The bottom-right shows the % of statistically significant comparisons (and $\hat{A}_{12}$) against AdMMO over 56 cases. The blue cells highlight the best in a case. The summarized raw results of all cases can be found at our repository: https://github.com/ideas-labo/admmo/blob/main/supp.pdf.

Optimizer	S_{100}	S_{200}	S_{300}	S_{400}	S_{100}	S_{200}	S_{300}	S_{400}	S_{100}	S_{200}	S_{300}	S_{400}	S_{100}	S_{200}	S_{300}	S_{400}
	MariaDB-Runtime				MariaDB-CPU				Storm-Throughtput				Storm-Latency			
IRACE	0.4774	0.3543	0.2813	0.2670	0.4933	0.3122	0.1673	0.1085	1.0000	0.6713	0.4944	0.3125	0.2317	0.1384	0.0932	0.0644
GA	0.5061	0.1138	0.0867	0.0702	0.2857	0.0715	0.0352	0.0091	0.1671	0.0741	0.0571	0.0429	0.0124	0.0024	0.0000	0.0000
RS	0.5172	0.3473	0.2468	0.1849	0.4280	0.2812	0.2207	0.1343	0.7768	0.4745	0.3254	0.2803	0.1792	0.1169	0.0922	0.0692
ParamILS	1.0000	0.5762	0.3904	0.1650	1.0000	0.7029	0.5779	0.5326	0.7240	0.4206	0.1829	0.0665	1.0000	0.7186	0.3986	0.2560
MMO	0.2166	0.1045	0.0712	0.0477	0.2761	0.0589	0.0268	0.0052	0.1173	0.0584	0.0435	0.0329	0.0427	0.0016	0.0000	0.0000
PMO	0.5124	0.1573	0.0834	0.0515	0.4505	0.3112	0.2365	0.1544	0.6520	0.1887	0.0914	0.0480	0.2813	0.0740	0.0216	0.0108
Flash	0.4543	0.2270	0.1815	0.1657	0.4979	0.0941	0.0033	0.0000	0.8734	0.0481	0.0213	0.0058	0.4423	0.0913	0.0913	0.0913
BOCA	0.5545	0.3208	0.2437	0.1924	0.4230	0.2808	0.2018	0.1897	0.3323	0.1725	0.0928	0.0485	0.2510	0.1325	0.0982	0.0814
SMAC	0.5573	0.3956	0.3155	0.2699	0.5258	0.3187	0.2214	0.1871	0.9983	0.6810	0.4461	0.2944	0.2198	0.1040	0.0768	0.0659
AdMMO	0.2122	0.0531	0.0093	0.0000	0.2823	0.1193	0.0654	0.0268	0.1237	0.0299	0.0173	0.0000	0.0416	0.0048	0.0000	0.0000
	VP9-Runtime				VP9-CPU				Keras-AUC				Keras-Time			
IRACE	0.1069	0.0665	0.0436	0.0296	0.0486	0.0216	0.0181	0.0162	0.7580	0.6093	0.4898	0.3819	0.0696	0.0424	0.0366	0.0309
GA	0.1315	0.0565	0.0246	0.0060	0.0537	0.0230	0.0120	0.0075	0.7959	0.6997	0.6501	0.6239	0.0082	0.0033	0.0022	0.0019
RS	0.1625	0.0625	0.0281	0.0240	0.0587	0.0310	0.0237	0.0188	0.8017	0.6706	0.5685	0.4927	0.0799	0.0387	0.0281	0.0242
ParamILS	1.0000	0.9901	0.9711	0.8899	1.0000	0.9833	0.9773	0.8772	1.0000	0.9913	0.9913	0.9854	1.0000	0.6231	0.4191	0.3367
MMO	0.1925	0.0512	0.0251	0.0004	0.0831	0.0531	0.0425	0.0351	0.8455	0.7638	0.6880	0.6122	0.0068	0.0030	0.0020	0.0013
PMO	0.1791	0.1021	0.0722	0.0519	0.0820	0.0585	0.0438	0.0371	0.2711	0.0000	0.0000	0.0000	0.0128	0.0023	0.0017	0.0013
Flash	0.1504	0.0647	0.0399	0.0162	0.0503	0.0290	0.0208	0.0164	0.9708	0.9708	0.9708	0.9708	0.0799	0.0202	0.0186	0.0186
BOCA	0.1627	0.0600	0.0459	0.0237	0.0748	0.0428	0.0264	0.0172	0.7580	0.6006	0.5510	0.4898	0.0479	0.0280	0.0232	0.0193
SMAC	0.1746	0.1004	0.0489	0.0259	0.0530	0.0285	0.0211	0.0190	0.7522	0.5685	0.5219	0.4490	0.0601	0.0391	0.0334	0.0293
AdMMO	0.0860	0.0451	0.0059	0.0000	0.0447	0.0193	0.0034	0.0000	0.7085	0.5569	0.4402	0.4169	0.0059	0.0012	0.0003	0.0000
	MongoDB-Runtime				MongoDB-CPU				x264-PSNR				x264-Energy			
IRACE	0.5129	0.2621	0.1853	0.1013	0.4202	0.1947	0.1079	0.0809	0.3351	0.3351	0.3351	0.3351	0.0077	0.0077	0.0077	0.0077
GA	0.5056	0.2795	0.1965	0.1041	0.4777	0.2792	0.1536	0.0519	0.1559	0.0487	0.0325	0.0179	0.0029	0.0018	0.0015	0.0015
RS	0.4724	0.2829	0.1900	0.1354	0.4741	0.2624	0.1837	0.0642	1.0000	1.0000	1.0000	1.0000	0.0961	0.0961	0.0961	0.0961
ParamILS	1.0000	1.0000	1.0000	1.0000	1.0000	0.9936	0.9936	0.9936	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
MMO	0.5541	0.3459	0.1746	0.0685	0.4844	0.2831	0.1862	0.1185	0.1918	0.0708	0.0214	0.0000	0.0044	0.0016	0.0014	0.0011
PMO	0.5140	0.2656	0.1681	0.0804	0.4675	0.2236	0.1475	0.0604	0.2510	0.1096	0.0574	0.0043	0.0051	0.0021	0.0011	0.0007
Flash	0.5821	0.3013	0.2191	0.1442	0.4782	0.2191	0.1154	0.0803	0.3351	0.3351	0.3351	0.3351	0.0077	0.0077	0.0077	0.0077
BOCA	0.6089	0.4453	0.2763	0.1810	0.4570	0.2776	0.1724	0.1157	0.3351	0.3351	0.3351	0.3351	0.0077	0.0077	0.0077	0.0077
SMAC	0.5709	0.3439	0.2264	0.1195	0.5364	0.2590	0.1605	0.0931	0.3351	0.3351	0.3351	0.3351	0.0077	0.0077	0.0077	0.0077
AdMMO	0.4332	0.1317	0.0361	0.0000	0.3829	0.1611	0.0269	0.0000	0.1424	0.0781	0.0194	0.0042	0.0032	0.0012	0.0001	0.0000
	LLVM-Runtime				LLVM-CPU				**vs. AdMMO**	**trivial**		**small**	**medium**		**large**	
IRACE	0.5210	0.4124	0.3804	0.3511	0.6465	0.4050	0.2831	0.2099	IRACE	7%		2%	2%		89%	
GA	0.0774	0.0000	0.0000	0.0000	0.4918	0.2615	0.1821	0.1608	GA	17%		9%	14%		60%	
RS	0.5998	0.4709	0.4278	0.3937	0.8072	0.5448	0.3880	0.2557	RS	2%		5%	2%		91%	
ParamILS	1.0000	0.9729	0.9510	0.9307	1.0000	0.9191	0.8688	0.8171	ParamILS	0%		0%	4%		96%	
MMO	0.0915	0.0000	0.0000	0.0000	0.4876	0.2396	0.1680	0.1164	MMO	16%		9%	21%		54%	
PMO	0.2613	0.0956	0.0276	0.0155	0.5450	0.3317	0.2534	0.2112	PMO	4%		5%	11%		80%	
Flash	0.4282	0.0000	0.0000	0.0000	0.4651	0.2760	0.1306	0.0277	Flash	14%		2%	4%		80%	
BOCA	0.3971	0.3287	0.2961	0.2720	0.7422	0.4786	0.3340	0.2130	BOCA	5%		2%	0%		93%	
SMAC	0.5058	0.3888	0.3204	0.2910	0.7112	0.4817	0.3210	0.2156	SMAC	5%		2%	2%		91%	
AdMMO	0.1156	0.0000	0.0000	0.0000	0.5216	0.2378	0.0984	0.0000	**% of significant cases vs. AdMMO over all 56 cases.**							

6.1.2 Findings. As can be seen from Table 2, it is clear that AdMMO achieves considerably better performance over the state-of-the-art optimizers, as it is ranked the best for 71% of the cases (40/56). In contrast to measurement-based optimizers such as IRACE, GA, RS, and ParamILS, the best improvement of AdMMO ranges from 21% to 100%. We also see that RS and ParamILS perform badly compared with the others, this is because the former has little heuristic to guide the tuning for better performance despite that it is good at handling local optima, while the latter suffers severely from the local optima issues due to the nature of its local search. Compared with the model-based optimizers, AdMMO again has superior results between 17% and 45% best improvements. Those optimizers appear to be also largely affected by the local optima issue as seen from cases of, e.g., on the *PSNR* for x264. When directly comparing with MMO, AdMMO also obtain much better results with up to 20% improvement. This evidences that the adaptive weight is indeed beneficial



(a) explored configurations (measurement-based optimizers)

(b) explored configurations (model-based optimizers)

(c) trajectory of w

(d) trajectory of p'

Fig. 7. Example run for system Storm. (a) and (b) are the projected landscape of explored configurations by all optimizers; (c) and (d) are the trajectories of w and the actual proportion of nondominated configurations p', respectively. The dashed line in (d) represents the expected proportion of nondominated configurations.

and necessary. With no surprise, PMO is much more inferior than AdMMO, except for one system-objective pair: i.e., *AUC* for Keras. Despite being rare, this is indeed possible: in this case, lower inference time often means better AUC (but not vice versa), hence optimizing the former can also benefit the latter. As such, the drawback of PMO we discussed in Section 2.4 would become blurred.

All the above results are concluded with high statistical significance, as can be seen at the bottom-right of Table 2: majority of the cases have significant comparison ($\hat{\mathbf{A}}_{12} \geq 0.56$ or $\hat{\mathbf{A}}_{12} \leq 0.44$ while $p\text{-}value < 0.05$) and exhibit medium to large effect size, i.e., $\hat{\mathbf{A}}_{12} \geq 0.64$ (or $\hat{\mathbf{A}}_{12} \leq 0.36$).

To take a closer look, Figure 7 shows all explored configurations together with how the w and p' change for a run. From Figures 7a and 7b, we see that AdMMO often successfully explores points around the promising regions, while the others can easily be trapped as some locally undesired areas. Unlike MMO, in Figures 7c and 7d, it is clear that AdMMO dynamically adapts the weights throughout the tuning and manages to maintain the proportion of nondominated configurations near a certain level, i.e., 30% in this case ($p = 0.3$).

> ***RQ1:*** *AdMMO is effective as it outperforms MMO and other optimizers in 71% cases with considerable improvements and high statistical significance.*

6.2 Efficiency

6.2.1 Method. To answer **RQ2**, we examine the efficiency of AdMMO by comparing how much less (or more) resources it needs to reach the best result obtained by the other optimizers. Similar to Chen and Li [Chen and Li 2021], we follow the steps below:

(1) Set a baseline, b, as the smallest number of measurements (up to 400) that one other optimizer needs to reach its best average (over 50 runs) of the target performance objective (says T).
(2) For AdMMO, find the smallest number of measurements, m, at which the mean of the target performance objective (over 50 runs) is at least the same as T.
(3) Following the metric used by Gao *et al.* [Gao et al. 2021], the speedup of AdMMO over its counterpart, i.e., $s = \frac{b}{m}$, is reported.

As such, for AdMMO to be efficient, we expect $s \geq 1$.

6.2.2 Findings. Figure 8 plots the results, from which we observe that compared with measurement-based optimizers (e.g., IRACE), AdMMO has $s \geq 1$ in the majority of the system-objective pairs (at least 12 out of 14), and it is at most of the time with $s > 1$ (at least 10 cases). Remarkably, the best

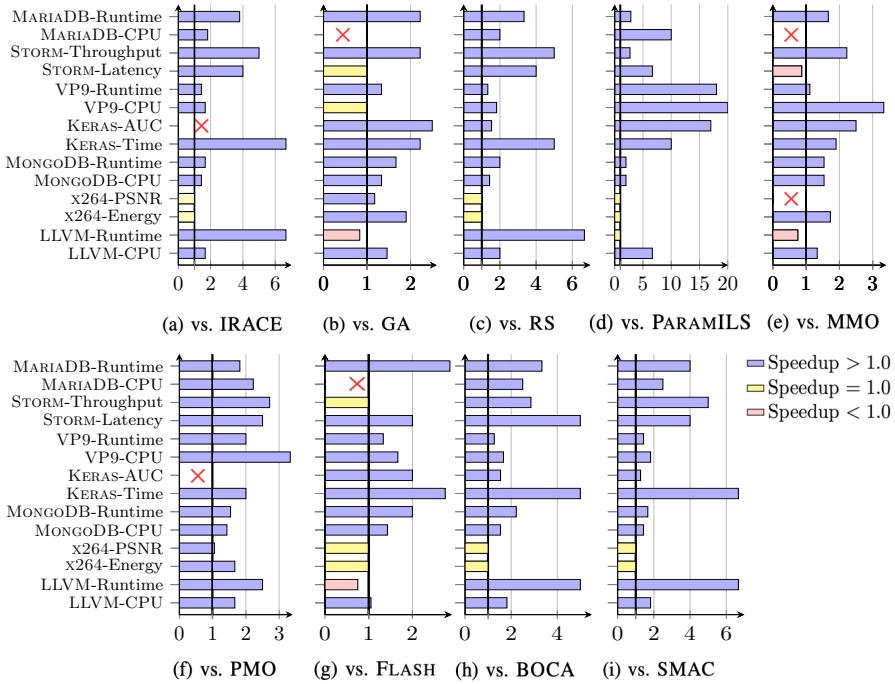


Fig. 8. The speedup (s) of AdMMO over the state-of-the-art optimizers on the budget of 400. ✕ denotes a case where AdMMO cannot achieve the same result when the budget runs out.

speedup ranges from 2.2× to 20× over the others. In contrast to the model-based Flash, BOCA, and SMAC, even without a surrogate, AdMMO can still obtain $s \geq 1$ for 12 to 14 pairs and at least 10 pairs of $s > 1$ with up to 6.7× speedup. AdMMO also significantly boosts MMO and PMO, achieving $s > 1$ on 10 and 13 out of 14 pairs, respectively. This means that the weight adaptation also enables more efficient resource consumption in the tuning.

> ***RQ2:*** *AdMMO is efficient, achieving $s > 1$ on at least 10 out of 14 system-objective pairs with the best speedup between 2.2× and 20× against the state-of-the-art optimizers.*

6.3 Ablation Analysis

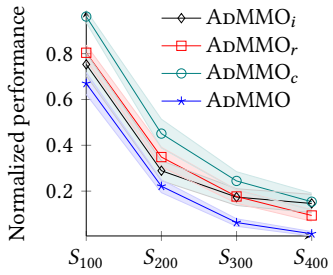
6.3.1 Method. To further study what parts in AdMMO can help to achieve the above results, in **RQ3**, we conduct an ablation analysis that converts AdMMO into the following variants:

- **AdMMO$_i$**: A variant that **i**ndistinguishes duplicate configurations in selection (Figure 6a).
- **AdMMO$_r$**: A variant that **r**emove all but one duplicates in selection (Figure 6b).
- **AdMMO$_c$**: A variant that **c**onstantly runs weight adaptation throughout the tuning, i.e., without the progressive trigger.

Their performance is compared with the original AdMMO.

6.3.2 Findings. From Figure 9, we see that all variants are clearly much inferior to AdMMO regardless of the budgets. This is also supported by the statistical test and $\hat{A}_{12}$, where the majority of the comparisons are statistically significant with medium to large effect sizes.

Interestingly, we observe that AdMMO$_i$ performs better than AdMMO$_r$ till around 200 budget size because, before this point, the number of duplicates might still be acceptable while it can be more harmful to preserve configurations dominated by those high-quality duplicate configurations



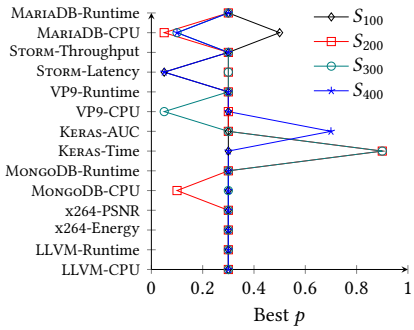
(a) overall performance

% of statistically significant cases vs. AdMMO

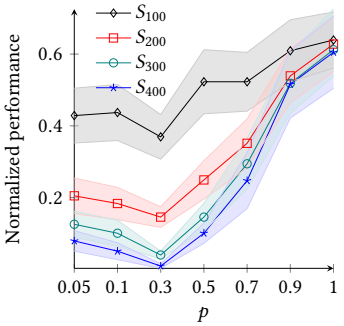
		AdMMO$_i$	AdMMO$_r$	AdMMO$_c$			AdMMO$_i$	AdMMO$_r$	AdMMO$_c$
S_{100}	trivial	14%	7%	0%	S_{300}	trivial	14%	21%	14%
	small	7%	21%	7%		small	7%	0%	14%
	medium	21%	29%	0%		medium	0%	21%	7%
	large	58%	43%	93%		large	79%	58%	65%
S_{200}	trivial	14%	22%	14%	S_{400}	trivial	14%	14%	14%
	small	22%	14%	7%		small	7%	0%	21%
	medium	14%	14%	7%		medium	0%	21%	0%
	large	50%	50%	72%		large	79%	65%	65%

(b) statistical significance

Fig. 9. Comparing AdMMO variants over 50 runs across all systems/objectives. (a) and (b) show the average/deviation of normalized target performance and the % of statistically significant comparisons (also classified based on $\hat{A}_{12}$) against AdMMO on 14 system-objective pairs, respectively.



(a) best p across systems-objective pairs



(b) impact of p on performance

Fig. 10. The role of p in AdMMO over 50 runs. (a) plots the best p on each system-objective pair; (b) shows the sensitivity of normalized target performance (mean and standard error) to p on all 56 cases.

that have been omitted, as we discussed in Section 4.3. Yet, AdMMO$_i$ worsens faster with a larger budget. This makes sense since the more measurements, the higher the likelihood of involving too many duplicates as the tuning converges, hence hindering the effectiveness of weight adaptation.

> ***RQ3:*** *Both progressive trigger and partial duplicate retention can greatly improve AdMMO across different systems, objectives and budgets.*

6.4 Sensitivity of AdMMO to Maintained Proportion of Nondominated Configurations p

6.4.1 Method. In **RQ4** we seek to empirically examine whether there exists a generally best p. To that end, we verify on different settings of p: $\{0.05, 0.1, 0.3, 0.5, 0.7, 0.9, 1\}$.

6.4.2 Findings. From Figure 10a. It is clear that except for a few cases, most of the time $p = 0.3$ tends to achieve the best performance regardless of the systems, objectives, and budgets. Additionally, from Figure 10b, we note that, for all budgets and across the systems/objectives, both too small (e.g., $p = 0.05$) or too large p (e.g., $p = 1$) can be detrimental, as the former suffers the local optima issues and the latter loses the discriminative power. In particular, we found that a too-large proportion is even more dangerous than a too-small one, as a small p still retains the selection pressure (despite being overpressurized) whereas a large p loses it completely.

RQ4: *Unlike the weight for which the best setting can largely vary case by case (recall Figure 3), maintaining a proportion of the nondominated configurations as around 30% ($p = 0.3$) can be overwhelmingly the best level according to all the cases in our study.*

7 THREATS TO VALIDITY

Threats to **internal validity** can be related to the tuning budget. To tackle this, we examine different budget sizes and assess the efficiency of utilizing resources. The parameters of optimizers can also harm internal validity, hence for the state-of-the-art optimizers, we set their values as identical to what have been fine-tuned in existing work. For AdMMO, we use the most pragmatic settings of the parameters that are less significant (e.g., T in the progressive trigger); most importantly, we confirm that there exists a generally best value for the most crucial parameter, i.e., p, for which we have used throughout the experiments. To mitigate bias, we repeated 50 experiment runs for each case. Indeed, we cannot completely rule out the impacts of some unusual parameter settings.

To reduce threats to **construct validity**, we compare the results of target performance objectives as that of Chen and Li [2021]. As recommended by Nair et al. [2020] and Gao et al. [2021], we also use the number of measurements (a language-independent feature) required to converge to the same result as an indicator of efficiency. To perform statistical validation, we leverage the non-parametric Wilcoxon test and $\hat{\mathbf{A}}_{12}$. The use of NSGA-II (and the operators) under AdMMO might be a threat, as it is merely a pragmatic choice, but it is not difficult to migrate AdMMO to the other multi-objective optimizers that are of the same family (i.e., compatible with nondominated sorting). We leave a more thorough study of diverse optimizers and operators in future work.

Threats to **external validity** can be raised from the subjects studied. We mitigated this by using seven systems that are of different scales and performance attributes, together with different budgets and nine state-of-the-art optimizers/models from diverse technical foundations. Nonetheless, if resources permit, we agree that using more systems and budgets may prove fruitful. It is worth noting that, in this work, we do not aim to optimize multiple performance objectives but use a tailored and improved idea/model of multi-objective search to solve a single objective SBSE problem. Indeed, there are cases where optimizing multiple performance objectives is desirable, yet we believe that this work serves as a first step and AdMMO can well be extended for those cases.

8 CONCLUSION AND FUTURE WORK

This paper presents a significant improvement on MMO for tuning software configuration, namely AdMMO. We contribute to a weight adaptation method that is capable of maintaining an unbiased proportion of nondominated configurations, together with a progressive trigger and a partial duplicate retention mechanism. Experiments on 14 system-objective pairs reveal that AdMMO:

- considerably outperforms the (improved) MMO and other optimizers for the tuning;
- and achieves so with a significant speedup in most cases while;
- maintaining around 30% of the nondominated configurations leads to the best outcome.

In future work, we hope to perform further analysis on the impact of the auxiliary performance objective chosen along with any characteristics, together with the role of different multi-objective optimizers and their operators. Understanding the suitability of AdMMO and the broader idea of multi-objectivization for other SBSE problems is also an interesting direction.

Data Availability: To promote open science, all source code, data, and supplementary materials can be accessed at our repository: https://github.com/ideas-labo/admmo.

ACKNOWLEDGMENTS

This work was supported by a UKRI Grant (10054084) and a NSFC Grant (62372084).

REFERENCES

Andrea Arcuri and Lionel C. Briand. 2011. A practical guide for using statistical tests to assess randomized algorithms in software engineering. In *ICSE'11: Proc. of the 33rd International Conference on Software Engineering*. ACM, 1–10.

Thomas Bartz-Beielstein, Frederik Rehbach, and Margarita Rebolledo. 2021. Tuning Algorithms for Stochastic Black-Box Optimization: State of the Art and Future Perspectives. *Black Box Optimization, Machine Learning, and No-Free Lunch Theorems* (2021), 67–108.

Babak Behzad, Huong Vu Thanh Luu, Joseph Huchette, Surendra Byna, Prabhat, Ruth A. Aydt, Quincey Koziol, and Marc Snir. 2013. Taming parallel I/O complexity with auto-tuning. In *International Conference for High Performance Computing, Networking, Storage and Analysis, SC'13, Denver, CO, USA - November 17 - 21, 2013*, William Gropp and Satoshi Matsuoka (Eds.). ACM, 68:1–68:12. https://doi.org/10.1145/2503210.2503278

James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. 2011. Algorithms for Hyper-Parameter Optimization. In *Advances in Neural Information Processing Systems 24: 25th Annual Conference on Neural Information Processing Systems 2011. Proceedings of a meeting held 12-14 December 2011, Granada, Spain*, John Shawe-Taylor, Richard S. Zemel, Peter L. Bartlett, Fernando C. N. Pereira, and Kilian Q. Weinberger (Eds.). 2546–2554. https://proceedings.neurips.cc/paper/2011/hash/86e8f7ab32cfd12577bc2619bc635690-Abstract.html

James Bergstra and Yoshua Bengio. 2012. Random Search for Hyper-Parameter Optimization. *J. Mach. Learn. Res.* 13 (2012), 281–305. http://dl.acm.org/citation.cfm?id=2188395

Nicola Beume, Boris Naujoks, and Michael T. M. Emmerich. 2007. SMS-EMOA: Multiobjective selection based on dominated hypervolume. *Eur. J. Oper. Res.* 181, 3 (2007), 1653–1669. https://doi.org/10.1016/J.EJOR.2006.08.008

Leslie Pérez Cáceres, Federico Pagnozzi, Alberto Franzin, and Thomas Stützle. 2017. Automatic Configuration of GCC Using Irace. In *Artificial Evolution - 13th International Conference, Évolution Artificielle, EA 2017, Paris, France, October 25-27, 2017, Revised Selected Papers (Lecture Notes in Computer Science, Vol. 10764)*, Evelyne Lutton, Pierrick Legrand, Pierre Parrend, Nicolas Monmarché, and Marc Schoenauer (Eds.). Springer, 202–216. https://doi.org/10.1007/978-3-319-78133-4_15

Junjie Chen, Ningxin Xu, Peiqi Chen, and Hongyu Zhang. 2021. Efficient Compiler Autotuning via Bayesian Optimization. In *43rd IEEE/ACM International Conference on Software Engineering, ICSE 2021, Madrid, Spain, 22-30 May 2021*. IEEE, 1198–1209. https://doi.org/10.1109/ICSE43902.2021.00110

Pengzhou Chen, Tao Chen, and Miqing Li. 2024. MMO: Meta Multi-Objectivization for Software Configuration Tuning. *IEEE Transactions on Software Engineering* (2024).

Tao Chen. 2022a. Lifelong Dynamic Optimization for Self-Adaptive Systems: Fact or Fiction?. In *IEEE International Conference on Software Analysis, Evolution and Reengineering, SANER 2022, Honolulu, HI, USA, March 15-18, 2022*. IEEE, 78–89. https://doi.org/10.1109/SANER53432.2022.00022

Tao Chen. 2022b. Planning Landscape Analysis for Self-Adaptive Systems. In *International Symposium on Software Engineering for Adaptive and Self-Managing Systems, SEAMS 2022, Pittsburgh, PA, USA, May 22-24, 2022*, Bradley R. Schmerl, Martina Maggio, and Javier Cámara (Eds.). ACM/IEEE, 84–90. https://doi.org/10.1145/3524844.3528060

Tao Chen and Rami Bahsoon. 2017a. Self-Adaptive and Online QoS Modeling for Cloud-Based Software Services. *IEEE Trans. Software Eng.* 43, 5 (2017), 453–475. https://doi.org/10.1109/TSE.2016.2608826

Tao Chen and Rami Bahsoon. 2017b. Self-Adaptive Trade-off Decision Making for Autoscaling Cloud-Based Services. *IEEE Trans. Serv. Comput.* 10, 4 (2017), 618–632. https://doi.org/10.1109/TSC.2015.2499770

Tao Chen, Ke Li, Rami Bahsoon, and Xin Yao. 2018. FEMOSAA: Feature Guided and Knee Driven Multi-Objective Optimization for Self-Adaptive Software. *ACM Transactions on Software Engineering and Methodology* 27, 2 (2018).

Tao Chen and Miqing Li. 2021. Multi-objectivizing software configuration tuning. In *ESEC/FSE '21: 29th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Athens, Greece, August 23-28, 2021*, Diomidis Spinellis, Georgios Gousios, Marsha Chechik, and Massimiliano Di Penta (Eds.). ACM, 453–465. https://doi.org/10.1145/3468264.3468555

Tao Chen and Miqing Li. 2023a. Do Performance Aspirations Matter for Guiding Software Configuration Tuning? An Empirical Investigation under Dual Performance Objectives. *ACM Trans. Softw. Eng. Methodol.* 32, 3 (2023), 68:1–68:41. https://doi.org/10.1145/3571853

Tao Chen and Miqing Li. 2023b. The Weights Can Be Harmful: Pareto Search versus Weighted Search in Multi-objective Search-based Software Engineering. *ACM Trans. Softw. Eng. Methodol.* 32, 1 (2023), 5:1–5:40. https://doi.org/10.1145/3514233

Tao Chen, Miqing Li, and Xin Yao. 2019. Standing on the shoulders of giants: Seeding search-based multi-objective optimization with prior knowledge for software service composition. *Inf. Softw. Technol.* 114 (2019), 155–175. https://doi.org/10.1016/j.infsof.2019.05.013

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

Pouria Derakhshanfar, Xavier Devroey, Andy Zaidman, Arie van Deursen, and Annibale Panichella. 2020. Good Things Come In Threes: Improving Search-based Crash Reproduction With Helper Objectives. In *35th IEEE/ACM International*

Conference on Automated Software Engineering (ASE'20).

Xiaoan Ding, Yi Liu, and Depei Qian. 2015. JellyFish: Online Performance Tuning with Adaptive Configuration and Elastic Container in Hadoop Yarn. In *21st IEEE International Conference on Parallel and Distributed Systems, ICPADS 2015, Melbourne, Australia, December 14-17, 2015*. IEEE Computer Society, 831–836. https://doi.org/10.1109/ICPADS.2015.112

Juan José Durillo and Antonio J. Nebro. 2011. jMetal: A Java framework for multi-objective optimization. *Adv. Eng. Softw.* 42, 10 (2011), 760–771. https://doi.org/10.1016/j.advengsoft.2011.05.014

Félix-Antoine Fortin and Marc Parizeau. 2013. Revisiting the NSGA-II crowding-distance computation. In *Genetic and Evolutionary Computation Conference, GECCO '13, Amsterdam, The Netherlands, July 6-10, 2013*, Christian Blum and Enrique Alba (Eds.). ACM, 623–630. https://doi.org/10.1145/2463372.2463456

Yanjie Gao, Yonghao Zhu, Hongyu Zhang, Haoxiang Lin, and Mao Yang. 2021. Resource-Guided Configuration Space Reduction for Deep Learning Models. In *43rd IEEE/ACM International Conference on Software Engineering, ICSE 2021, Madrid, Spain, 22-30 May 2021*. IEEE, 175–187. https://doi.org/10.1109/ICSE43902.2021.00028

Jingzhi Gong and Tao Chen. 2022. Does Configuration Encoding Matter in Learning Software Performance? An Empirical Study on Encoding Schemes. In *19th IEEE/ACM International Conference on Mining Software Repositories, MSR 2022, Pittsburgh, PA, USA, May 23-24, 2022*. ACM, 482–494. https://doi.org/10.1145/3524842.3528431

Jingzhi Gong and Tao Chen. 2023. Predicting Software Performance with Divide-and-Learn. In *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/FSE 2023, San Francisco, CA, USA, December 3-9, 2023*, Satish Chandra, Kelly Blincoe, and Paolo Tonella (Eds.). ACM, 858–870. https://doi.org/10.1145/3611643.3616334

Jingzhi Gong and Tao Chen. 2024. Predicting Configuration Performance in Multiple Environments with Sequential Meta-Learning. *FSE'24: Proceedings of the ACM on Software Engineering (PACMSE)* 1, FSE. https://doi.org/10.1145/3643743

Xue Han and Tingting Yu. 2016. An Empirical Study on Performance Bugs for Highly Configurable Software Systems. In *Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, ESEM 2016, Ciudad Real, Spain, September 8-9, 2016*. ACM, 23:1–23:10. https://doi.org/10.1145/2961111.2962602

Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. 2011. Sequential Model-Based Optimization for General Algorithm Configuration. In *LION5: Proc. of the 5th International Conference Learning and Intelligent Optimization (Lecture Notes in Computer Science, Vol. 6683)*. Springer, 507–523.

Frank Hutter, Holger H. Hoos, Kevin Leyton-Brown, and Thomas Stützle. 2009. ParamILS: An Automatic Algorithm Configuration Framework. *J. Artif. Intell. Res.* 36 (2009), 267–306. https://doi.org/10.1613/jair.2861

Pooyan Jamshidi and Giuliano Casale. 2016. An Uncertainty-Aware Approach to Optimal Configuration of Stream Processing Systems. In *24th IEEE International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems, MASCOTS 2016, London, United Kingdom, September 19-21, 2016*. IEEE Computer Society, 39–48.

Pooyan Jamshidi, Miguel Velez, Christian Kästner, and Norbert Siegmund. 2018. Learning to sample: exploiting similarities across environments to learn performance models for configurable systems. In *Proceedings of the 2018 ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/SIGSOFT FSE 2018, Lake Buena Vista, FL, USA, November 04-09, 2018*, Gary T. Leavens, Alessandro Garcia, and Corina S. Pasareanu (Eds.). ACM, 71–82. https://doi.org/10.1145/3236024.3236074

Ke Li, Zilin Xiang, Tao Chen, and Kay Chen Tan. 2020a. BiLO-CPDP: Bi-Level Programming for Automated Model Discovery in Cross-Project Defect Prediction. In *35th IEEE/ACM International Conference on Automated Software Engineering, ASE 2020, Melbourne, Australia, September 21-25, 2020*. IEEE, 573–584. https://doi.org/10.1145/3324884.3416617

Ke Li, Zilin Xiang, Tao Chen, Shuo Wang, and Kay Chen Tan. 2020b. Understanding the automated parameter optimization on transfer learning for cross-project defect prediction: an empirical study. In *ICSE '20: 42nd International Conference on Software Engineering, Seoul, South Korea, 27 June - 19 July, 2020*, Gregg Rothermel and Doo-Hwan Bae (Eds.). ACM, 566–577. https://doi.org/10.1145/3377811.3380360

Miqing Li, Tao Chen, and Xin Yao. 2022. How to Evaluate Solutions in Pareto-Based Search-Based Software Engineering: A Critical Review and Methodological Guidance. *IEEE Trans. Software Eng.* 48, 5 (2022), 1771–1799. https://doi.org/10.1109/TSE.2020.3036108

Min Li, Liangzhao Zeng, Shicong Meng, Jian Tan, Li Zhang, Ali Raza Butt, and Nicholas C. Fuller. 2014. MRONLINE: MapReduce online performance tuning. In *The 23rd International Symposium on High-Performance Parallel and Distributed Computing, HPDC'14, Vancouver, BC, Canada - June 23 - 27, 2014*, Beth Plale, Matei Ripeanu, Franck Cappello, and Dongyan Xu (Eds.). ACM, 165–176. https://doi.org/10.1145/2600212.2600229

Manuel López-Ibánez, Jérémie Dubois-Lacoste, Leslie Pérez Cáceres, Mauro Birattari, and Thomas Stützle. 2016. The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives* 3 (2016), 43–58.

Martin Lukasiewycz, Michael Glaß, Felix Reimann, and Jürgen Teich. 2011. Opt4J: a modular framework for meta-heuristic optimization. In *13th Annual Genetic and Evolutionary Computation Conference, GECCO 2011, Proceedings, Dublin, Ireland, July 12-16, 2011*, Natalio Krasnogor and Pier Luca Lanzi (Eds.). ACM, 1723–1730. https://doi.org/10.1145/2001576.2001808

Pedro Mendes, Maria Casimiro, Paolo Romano, and David Garlan. 2020. TrimTuner: Efficient Optimization of Machine Learning Jobs in the Cloud via Sub-Sampling. In *28th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems, MASCOTS 2020, Nice, France, November 17-19, 2020*. IEEE, 1–8. https://doi.org/10.1109/MASCOTS50786.2020.9285971

Mohamed Wiem Mkaouer, Marouane Kessentini, Slim Bechikh, and Mel Ó Cinnéide. 2014. A Robust Multi-objective Approach for Software Refactoring under Uncertainty. In *Search-Based Software Engineering - 6th International Symposium, SSBSE 2014, Fortaleza, Brazil, August 26-29, 2014. Proceedings (Lecture Notes in Computer Science, Vol. 8636)*, Claire Le Goues and Shin Yoo (Eds.). Springer, 168–183. https://doi.org/10.1007/978-3-319-09940-8_12

Vivek Nair, Zhe Yu, Tim Menzies, Norbert Siegmund, and Sven Apel. 2020. Finding faster configurations using FLASH. *IEEE Transactions on Software Engineering* 46, 7 (2020).

Andrew Jhon Scott and M Knott. 1974. A cluster analysis method for grouping means in the analysis of variance. *Biometrics* (1974), 507–512.

Arman Shahbazian, Suhrid Karthik, Yuriy Brun, and Nenad Medvidovic. 2020. eQual: informing early design decisions. In *ESEC/FSE '20: 28th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Virtual Event, USA, November 8-13, 2020*, Prem Devanbu, Myra B. Cohen, and Thomas Zimmermann (Eds.). ACM, 1039–1051. https://doi.org/10.1145/3368089.3409749

Norbert Siegmund, Sergiy S. Kolesnikov, Christian Kästner, Sven Apel, Don S. Batory, Marko Rosenmüller, and Gunter Saake. 2012. Predicting performance via automated feature-interaction detection. In *34th International Conference on Software Engineering, ICSE 2012, June 2-9, 2012, Zurich, Switzerland*, Martin Glinz, Gail C. Murphy, and Mauro Pezzè (Eds.). IEEE Computer Society, 167–177. https://doi.org/10.1109/ICSE.2012.6227196

Moisés Silva-Muñoz, Alberto Franzin, and Hugues Bersini. 2021. Automatic configuration of the Cassandra database using irace. *PeerJ Comput. Sci.* 7 (2021), e634. https://doi.org/10.7717/peerj-cs.634

Mozhan Soltani, Pouria Derakhshanfar, Annibale Panichella, Xavier Devroey, Andy Zaidman, and Arie van Deursen. 2018. Single-objective Versus Multi-objectivized Optimization for Evolutionary Crash Reproduction. In *Search-Based Software Engineering - 10th International Symposium, SSBSE 2018, Montpellier, France, September 8-9, 2018, Proceedings (Lecture Notes in Computer Science, Vol. 11036)*, Thelma Elita Colanzi and Phil McMinn (Eds.). Springer, 325–340. https://doi.org/10.1007/978-3-319-99241-9_18

Pavel Valov, Jean-Christophe Petkovich, Jianmei Guo, Sebastian Fischmeister, and Krzysztof Czarnecki. 2017. Transferring Performance Prediction Models Across Different Hardware Platforms. In *Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering, ICPE 2017, L'Aquila, Italy, April 22-26, 2017*, Walter Binder, Vittorio Cortellessa, Anne Koziolek, Evgenia Smirni, and Meikel Poess (Eds.). ACM, 39–50. https://doi.org/10.1145/3030207.3030216

András Vargha and Harold D. Delaney. 2000. A Critique and Improvement of the CL Common Language Effect Size Statistics of McGraw and Wong.

Frank Wilcoxon. 1945. Individual Comparisons by Ranking Methods.

Shengxiang Yang, Miqing Li, Xiaohui Liu, and Jinhua Zheng. 2013. A Grid-Based Evolutionary Algorithm for Many-Objective Optimization. *IEEE Trans. Evol. Comput.* 17, 5 (2013), 721–736. https://doi.org/10.1109/TEVC.2012.2227145

Yuan Yuan and Wolfgang Banzhaf. 2020. ARJA: Automated Repair of Java Programs via Multi-Objective Genetic Programming. *IEEE Trans. Software Eng.* 46, 10 (2020), 1040–1067. https://doi.org/10.1109/TSE.2018.2874648

Yuqing Zhu, Jianxun Liu, Mengying Guo, Yungang Bao, Wenlong Ma, Zhuoyue Liu, Kunpeng Song, and Yingchun Yang. 2017. BestConfig: tapping the performance potential of systems via automatic configuration tuning. In *Proceedings of the 2017 Symposium on Cloud Computing, SoCC 2017, Santa Clara, CA, USA, September 24-27, 2017*. ACM, 338–350. https://doi.org/10.1145/3127479.3128605

Eckart Zitzler and Simon Künzli. 2004. Indicator-Based Selection in Multiobjective Search. In *Parallel Problem Solving from Nature - PPSN VIII, 8th International Conference, Birmingham, UK, September 18-22, 2004, Proceedings (Lecture Notes in Computer Science, Vol. 3242)*, Xin Yao, Edmund K. Burke, José Antonio Lozano, Jim Smith, Juan Julián Merelo Guervós, John A. Bullinaria, Jonathan E. Rowe, Peter Tiňo, Ata Kabán, and Hans-Paul Schwefel (Eds.). Springer, 832–842. https://doi.org/10.1007/978-3-540-30217-9_84

Marcela Zuluaga, Andreas Krause, and Markus Püschel. 2016. e-PAL: An Active Learning Approach to the Multi-Objective Optimization Problem. *J. Mach. Learn. Res.* 17 (2016), 104:1–104:32. http://jmlr.org/papers/v17/15-047.html

Received 2023-09-27; accepted 2024-01-23