MMO

Chen, Pengzhou; Chen, Tao; Li, Miqing

DOI:
10.1109/TSE.2024.3388910

License:
Creative Commons: Attribution (CC BY)

Document Version
Peer reviewed version

Citation for published version (Harvard):
Chen, P, Chen, T & Li, M 2024, 'MMO: Meta multi-objectivization for software configuration tuning', *IEEE Transactions on Software Engineering*. https://doi.org/10.1109/TSE.2024.3388910

MMO: Meta Multi-Objectivization for Software Configuration Tuning

Pengzhou Chen, Tao Chen, *Member, IEEE,* and Miqing Li, *Senior Member, IEEE*

Abstract—Software configuration tuning is essential for optimizing a given performance objective (e.g., minimizing latency). Yet, due to the software's intrinsically complex configuration landscape and expensive measurement, there has been a rather mild success, particularly in preventing the search from being trapped in local optima. To address this issue, in this paper we take a different perspective. Instead of focusing on improving the optimizer, we work on the level of optimization model and propose a meta multi-objectivization (MMO) model that considers an auxiliary performance objective (e.g., throughput in addition to latency). What makes this model distinct is that we do not optimize the auxiliary performance objective, but rather use it to make similarly-performing while different configurations less comparable (i.e. Pareto nondominated to each other), thus preventing the search from being trapped in local optima. Importantly, by designing a new normalization method, we show how to effectively use the MMO model without worrying about its weight—the only yet highly sensitive parameter that can affect its effectiveness. Experiments on 22 cases from 11 real-world software systems/environments confirm that our MMO model with the new normalization performs better than its state-of-the-art single-objective counterparts on 82% cases while achieving up to $2.09\times$ speedup. For 68% of the cases, the new normalization also enables the MMO model to outperform the instance when using it with the normalization from our prior FSE work under pre-tuned best weights, saving a great amount of resources which would be otherwise necessary to find a good weight. We also demonstrate that the MMO model with the new normalization can consolidate recent model-based tuning tools on 68% of the cases with up to $1.22\times$ speedup in general.

Index Terms—Configuration tuning, performance optimization, search-based software engineering, multi-objectivization

1 Introduction

Many software systems are highly configurable, such that the configuration options can be flexibly adjusted for performance, including database systems, machine learning systems, and cloud systems, to name a few. For example, Apache Storm, a stream processing system, can be tuned by changing some key configuration options such as `splitters`. However, a daunting number of configuration options will inevitably introduce a high risk of inappropriate or even poor software configurations set by software engineers. It has been reported that 59% of the software performance issues worldwide are related to ill-suited configuration rather than code [43]. In 2017-2018, configuration-related performance issues costed at least 400,000 USD per hour for 50% of the software companies[1].

Indeed, adjusting the configurations will affect the outcomes of different performance attributes, such as latency, throughput, and CPU load [80], [26], [70], [25], [23], [24]. However, there are many cases wherein only the optimization of a single performance attribute is of interest, whose minimization/maximization serves as a sole performance objective in consideration. For example, in the finance sector, a millisecond decrease in the trade delay may boost a high-speed firm's earnings by about 100 million USD per year [90]. Another example is related to the machine learning systems deployed by large organizations (e.g., GPT-4 [72]), or those in the health care domain [3], where the concern is mainly on the accuracy, while caring little about the overhead/resource incurred for training. This has been well-echoed from the literature on software configuration tuning, in the majority of which only a single performance attribute is considered at a time [8], [99], [71], [95], [62], [7], [61], [60].

Despite only a single performance attribute being of concern, such an optimization scenario is not easy to deal with for any optimizer that tunes the software configuration. This is because

1) The configurable systems involve a daunting number of configuration options with complex interactions, rendering a black-box to the software engineers [97], [18], [21], [22].
2) The measurement of each configuration through running the software system is often expensive [51], hence exhaustively exploring every configuration is unrealistic.
3) There is generally a high degree of sparsity in the configurable software systems [70], [19], [20], i.e., similar configurations can also have radically different performance.

The last characteristic poses a particular challenge to the automatic software configuration tuning in finding the optimal configuration (performance), because firstly different configurations may achieve locally good, but globally inadequate performance (e.g., local optima); and secondly, the landscape of a (local) optimum's neighborhood can be steep and rugged—if the tuning is trapped in a local optimum, it may be hard to escape from it as their neighboring configurations often perform significantly worse than it. As an example, Figure 1 shows the projected configuration landscape for Apache Storm (2 out of 6 configuration options), where it can be clearly seen that even with this simplified version,

- *Tao Chen is the corresponding author (t.chen@bham.ac.uk).*
- *Pengzhou Chen is with the School of Computer Science and Engineering, University of Electronic Science and Technology of China, China.*
- *Tao Chen and Miqing Li are with the School of Computer Science, University of Birmingham, UK.*

1. https://www.evolven.com/blog/downtime-outages-and-failures-understanding-their-true-costs.html.

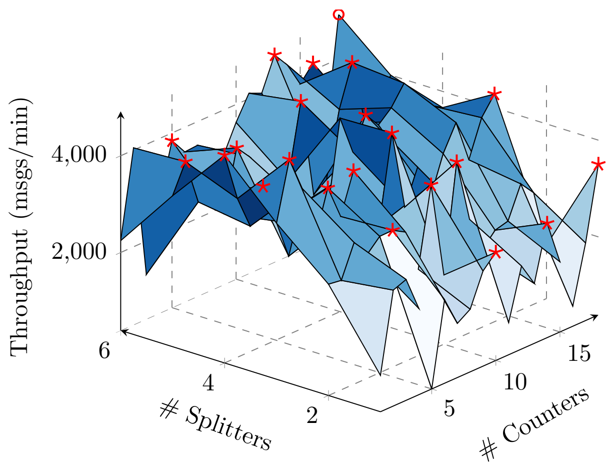


Fig. 1: A projected landscape of the performance objective *Throughput* with respect to configuration options `Splitters` and `Counters` for STORM under the WORDCOUNT benchmark. ○ is the global optimum and ⋆ denotes the local optima of throughput that an optimizer needs to escape from.

the landscape is rather rugged and contains steep "local optimum traps", resulting in significant difficulty in the tuning.

In light of the above challenges, a number of optimizers from the Search-Based Software Engineering (SBSE) paradigm have been presented, such as random search [8], [99], [71], hill climbing [95], [62], genetic algorithm [7], [80], [78], [84], and simulated annealing [37], [42]. To seek the global optimum (best performance of the concerned performance attribute) while avoiding being trapped in local optima, such methods focus on the "internal" components of the optimizer. They work on designing novel search operators (i.e., the way to change the configuration structure, for example, increasing the neighbourhood size of randomly mutated configurations [71]), or developing various search strategies (i.e., the way to balance exploration and exploitation, for example, restarting the search in hill climbing [95]). However, a common limitation of such single-objective optimizers is that the goal to find the global optimum is "less oriented" as there is no clear "incentive" to encourage them to traverse the wide search space and locating as many local optima as possible, thus finding the best one in a resource-efficient manner.

To better mitigate the local optima, in this paper and our prior FSE'21 work [27] (we call it FSE work thereafter), we tackle this software configuration tuning problem from a different perspective. In contrast to the effort made by the existing works on the development of the optimizer, we work on the optimization model, i.e., the "external" part of an optimizer. This is achieved by proposing a meta multi-objectivization (MMO) model for this single-objective problem, to help the search avoid being trapped in local optima and progressively explore the entire objective space.

In a nutshell, MMO seeks to optimize two **meta-objectives**, each of which has two components. The first component of both meta-objectives is the target performance objective (e.g., latency), thereby only those configurations that perform well on the target objective being in favor. The second component, which is related to the other given auxiliary performance objective (e.g., throughput), is a completely conflicting term for the two meta-objectives. The reason for this design is that we hope to keep the target performance objective as a primary term in the model to preserve the tendency towards its optimality, but at the same time, we want the configurations with different values on the auxiliary performance objective to be incomparable. We are not interested in minimizing/maximizing the auxiliary performance objective since we do not know which value of it can lead to the best result on the target performance objective, but we wish to keep a good amount of configurations with diverse values of the auxiliary performance objective in the search, thus not being trapped in local optima (we will elaborate on this in Section 3). The contributions from both this work and the FSE work are:

- Unlike existing work for the software configuration tuning which puts effort on the "internal part" of the optimization (i.e., improving the search operators of various optimizers), we work on the "external part"—multi-objectivizing this single-objective optimization scenario.
- We present a meta multi-objectivization model, MMO, as opposed to the existing multi-objectivization model considered in other SBSE scenarios which directly optimizes the target and auxiliary objectives simultaneously (referred to as plain multi-objectivization or PMO). We show, analytically and experimentally, why MMO is more suitable than PMO for software configuration tuning.

However, MMO requires a weight parameter to aggregate the target objective component and the auxiliary objective component. It is a critical parameter to balance searching for a good target performance objective value and maintaining diverse auxiliary performance objective values, requiring fine-tuning from the software engineers for every configurable software/environment, as done in our FSE work [27]. This, if done inappropriately, could lead to poor outcomes, as we will show in Section 3.5. Yet, since the measurement of configurations is often expensive, finding the best weight in a case-by-case manner is not always realistic, which is a major threat to the applicability of the MMO model.

Therefore, in this paper, we also tackle this unwelcome issue. We show why the weight can be a highly sensitive parameter in the MMO model and propose a way to make the model weight-free without compromising the result. This is achieved by presenting a new normalization method, which is simple, but works very well—it leads to results that are even better than those of the FSE work under its best-tuned weight [27] for the majority of the problems. To sum up, the unique contributions of this paper are:

- A sound and formal analysis of the principle behind MMO, derived from the perspective of geometric transformation in the performance objective space, that explains its intention and what role the parameter w and the normalization play therein. This then enables us to formally reflect on the limitation posed by the MMO model design proposed in the FSE work.
- Drawing on insights from the analysis, we design a new normalization method as part of the MMO model,

capturing the bounds of both performance objectives adaptively. This allows us to keep the strengths and characteristics of the MMO model while removing the weight (i.e., setting $w = 1$ for all cases).

- An extensive evaluation that expands to 11 systems/environments that are of very different domains. Since a system comes with two performance objectives, each of these is used in turn as the target performance objective, leading to 22 cases. Under these cases, we compare MMO model using the new normalization with the PMO model and four single-objective counterparts, as well as with the MMO model using the normalization from the FSE work.
- An investigation on how our MMO model with the new normalization can consolidate FLASH [70] and BOCA [17], which are state-of-the-art model-based tuning methods for software configuration tuning.

Our experiment results are encouraging: we show that the MMO model with the new normalization achieves better results over the best single-objective counterpart and PMO (on 18 and 20 out of 22 cases wherein 14 and 15 of them are considerably better, respectively), while being much more resource-efficient overall (with up to $2.09\times$ speedup over the single-objective optimizers and use significantly less resource than that of the PMO). In contrast to using the MMO model with the normalization from FSE work under its best weight, the MMO model with the new normalization shows better results on 15 cases (7 of which are significant) and competitive resource efficiency. Notably, this is achieved without the need of setting the weight, which can be undesirable as in 13 out of 22 cases, it requires at least 50% of the search budget as the extra resource to identify the best weight. The MMO model with the new normalization can also consolidate the model-based tuning methods like FLASH and BOCA: with minimal code change, both can be improved for 15 out of 22 cases (with 12 or 13 cases of statistically significant improvement) while having a $1.22\times$ and $1.06\times$ speedup in general, respectively.

To promote the open-science practice, a GitHub repository that contains all source code and data in this work can be accessed at: https://github.com/ideas-labo/mmo.

The rest of this paper is organized as follows. Section 2 introduces some background information. Section 3 elaborates on the design of the MMO and PMO model, as well as why and how we design the new normalization. Section 4 presents our experiment methodology, followed by a detailed discussion of the results in Section 5. Section 6 delineates how to apply MMO in practical software engineering scenarios. The threats to validity are discussed in Section 7. Sections 8 and 9 analyze the related work and conclude the paper, respectively.

2 PRELIMINARIES

In this section, we describe the necessary background information and context for this work.

2.1 Software Configuration Tuning Problem

A configurable software system often comes with a set of critical configuration options such that the ith option is denoted as x_i, which can be either a binary or integer variable, where n is the total number of options. The search space, $\boldsymbol{\mathcal{X}}$, is the Cartesian product of the possible values for all the x_i. Formally, when only a single performance concern is of interest (such as latency, throughput, or accuracy), the goal of software configuration tuning is to achieve[2]:

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

where $\boldsymbol{x} = (x_1, x_2, ..., x_n)$. This is a classic *single-objective optimization model* and the measurement of f is entirely case-dependent according to the target software and the corresponding performance attribute; thus we make no assumption about its characteristics.

2.2 Multi-objectivization

Multi-objectivization is the method of transforming a single-objective optimization problem into a multi-objective one, in order to make the search easier to find the global optimum. It can be realized by adding a new objective (or several objectives) to the original objective or replacing the original objective with a set of objectives. The motivation is that since in a complex problem landscape, the search may get trapped in local optima when considering the original objective (due to the total order relation between solutions with respect to that objective), considering multiple objectives may make similarly-performed solutions incomparable (i.e., Pareto nondominated to each other), thus helping the search jump out of local optima [55].

Two solutions being Pareto nondominated means that one is better than the other on some objective and worse on some other objective. Formally, for two solutions $\boldsymbol{x}$ and $\boldsymbol{y}$, we call $\boldsymbol{x}$ and $\boldsymbol{y}$ nondominated to each other if $\boldsymbol{x} \nprec \mathbf{y} \wedge \boldsymbol{y} \nprec \boldsymbol{x}$, where $\nprec$ is the negation of "to Pareto dominate" ($\prec$), the superiority relation between solutions for multi-objective optimization. That is, considering a minimization problem with m objectives, $\boldsymbol{x}$ is said to *(Pareto) dominate* $\boldsymbol{y}$ (denoted as $\boldsymbol{x} \prec \boldsymbol{y}$) if $f_i(\boldsymbol{x}) \leq f_i(\boldsymbol{y})$ for $1 \leq i \leq m$ and there exists at least one objective j on which $f_j(\boldsymbol{x}) < f_j(\boldsymbol{y})$. Pareto dominance is a partial order relation, and thus there typically exist multiple optimal solutions in multi-objective optimization. For a solution set $\boldsymbol{X}$, a solution $\boldsymbol{x} \in \boldsymbol{X}$ is called *Pareto optimal* to $\boldsymbol{X}$ if there is no solution $\in \boldsymbol{X}$ that dominates $\boldsymbol{x}$. When $\boldsymbol{X}$ is the collection of all feasible solutions for a multi-objective problem, $\boldsymbol{x}$ becomes an optimal solution to the problem, and the set of all Pareto optimal solutions of the problem is called its *Pareto optimal set*.

Multi-objectivization is not uncommon in the modern optimization realm, particularly to the evolutionary computation community [55], [16], [50], [87], [88]. To tackle various challenging single-objective optimization problems, researchers put much effort in introducing/designing additional objectives, e.g., creating sub-problems (sub-objectives) of the original objective [55], converting the constraints into an additional objective [16], constructing similar adjustable objectives [50], considering one of the decision variables [87], or even adding a man-made less relevant objective function [88].

2. Without loss of generality, we assume the performance objective to be minimized.

3 Multi-objectivization for Software Configuration Tuning

In this section, we present the designs of the multi-objectivization models and how they are derived from the key properties in software configuration tuning.

3.1 Properties in Configuration Tuning

We observed that, in general, software configuration tuning bears the following properties.

Property 1: As shown in Figure 1 and what has already been reported [70], [51], [28], [29], the configuration landscape of different performance objectives for most configurable software systems is rather rugged with numerous local optima at varying slopes. Therefore the tuning, once the search is trapped at a local optimum, would be difficult to progress. This is because all the surrounding configurations of a local optimum are significantly inferior to it, and the search focus would have no much drive to move away from that local optimum (if only the concerned performance attribute is used to guide the search). As a result, a good optimization model has some additional "tricks" to avoid comparing configurations solely based on a single performance attribute.

Property 2: A single measurement of configuration is often expensive. For example, Valov *et al.* [92] reported that sampling all values of 11 configuration options for X264 needs 1,536 hours. This means that the resource (search budget) in software configuration tuning is highly valuable, hence utilizing them efficiently is critical.

Property 3: The correlation between different performance attributes is often uncertain, as different configurations may have different effects on distinct attributes. We observed that the configurations may achieve extremely good or bad performance on one while having similarly good results on the other, as illustrated in Figure 2. Taking the system STORM with ROLLINGSORT benchmark (denoted STORM/RS) from Figure 2 (left) as an example, suppose that in a multi-threaded and multi-core environment with 100 successful messages, if a configuration $\boldsymbol{A}$ enables each of these messages to be processed at 30ms, then the latency and throughput are $\frac{100\times30}{100} = 30$ms and $\frac{100}{30} = 3.33$ msgs/ms, respectively. In contrast, another configuration $\boldsymbol{B}$ may restrict the parallelism (e.g., lower `spout_num`), hence there could be 50 messages processed at 20ms each[3] while the other 50 are handled at 40ms each (including 20ms queuing time due to reduced parallelism). Here, the latency remains at $\frac{50\times20+50\times40}{100} = 30$ms but the throughput is changed to $\frac{100}{40} = 2.5$ msgs/ms, which is a 25% drop. Therefore, we should neither presume a strict conflict nor a harmonic correlation between the performance attributes.

Clearly, a good optimization model for software configuration tuning needs to take the above properties into account.

3.2 Plain Multi-Objectivization (PMO) Model

A straightforward idea to perform multi-objectivization is to add an auxiliary objective to optimize, along with optimizing the target performance objective. This is what has been commonly used in SBSE scenarios (e.g., [33], [69], [15], [1]). That PMO model can be formulated as:

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

where $f_t(\boldsymbol{x})$ denotes the target performance objective (i.e., the concerned one) and $f_a(\boldsymbol{x})$ denotes the auxiliary performance objective[4].

Putting it in the context of software configuration tuning, the PMO model may cover **Property 1**, because the natural Pareto relation with respect to the two objectives ensures that the target performance objective is no longer a sole indicator to guide the search. However, it does not fit **Property 2** as PMO additionally optimizes the auxiliary performance objective. As such, configurations that perform well on the auxiliary performance objective but poorly on the target performance objective are still regarded as optimal in PMO, despite being meaningless to the original problem. This can result in a significant waste of resources. In addition, PMO does not consider **Property 3** as it often assumes conflicting correlation between the two objectives [68], [33], which is hard to assure in software configuration tuning.

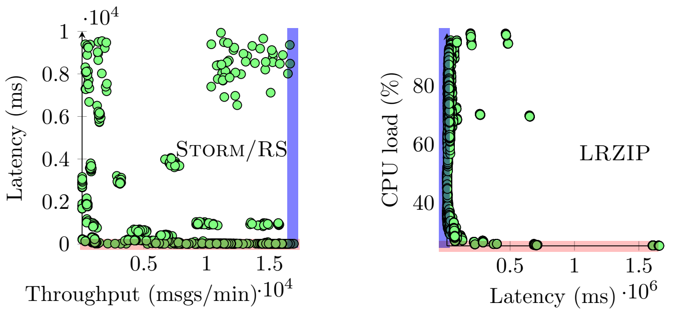


Fig. 2: Measured configurations for system STORM/RS and LRZIP. The points that **Property 3** refers to are highlighted: very good or bad results on one performance objective can both correspond to similarly good values on the other.

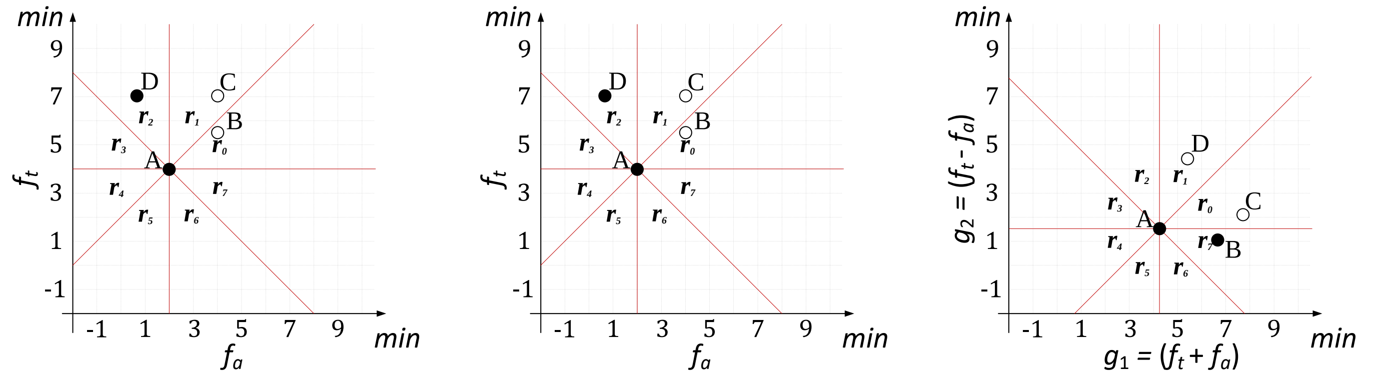
3.3 Meta Multi-Objectivization (MMO) Model

Unlike PMO, our meta multi-objectivization (MMO) model creates two meta-objectives based on the performance attributes. The aim is to drive the search towards the optimum of the target performance objective and at the same time, not to be trapped in local optima. In particular, we want to achieve two goals:

— **Goal 1:** optimizing the target performance objective still plays a primary role, thus no resource waste on, for example, optimizing the auxiliary one (this fits in **Property 2**);
— **Goal 2:** but those with different values of the auxiliary performance objective are more likely to be incomparable (i.e., Pareto nondominated), hence the search would

3. The relief of peak CPU load could allow the process of each message faster.

4. Without loss of generality, we use the minimization form of the performance objectives; the maximization ones can be trivially converted, e.g., by multiplying −1.



(a) Original space (b) Scaling at $w = 1$ in MMO (c) Rotation/dilation at $w = 1$ in MMO

Fig. 3: An illustration of the rotation effect in MMO. $\boldsymbol{A}$, $\boldsymbol{B}$, $\boldsymbol{C}$, and $\boldsymbol{D}$ are four configurations with the auxiliary and target performance objective values as $(2, 4)$, $(4, 5.5)$, $(4, 7)$, and $(0.75, 7)$, respectively. The auxiliary and target performance objectives are to be minimized if they are both of concern. $\circ$ denotes the nondominated configurations while $\bullet$ means those that are dominated by at least one other within the corresponding current space.

not be trapped in local optima (this relates to **Properties 1** and **3**).

Formally, the MMO model with two meta-objectives $g_1(\boldsymbol{x})$ and $g_2(\boldsymbol{x})$ is constructed as[5]:

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

whereby each of the two meta-objectives shares the same target performance objective $f_t(\boldsymbol{x})$, but differs (effectively being opposite) regarding the auxiliary performance objective $f_a(\boldsymbol{x})$. The auxiliary objective can be a readily available one and whose result is of no interest (e.g., throughput or CPU load, in addition to latency). The weight w is a critical parameter that balances the target and auxiliary performance objectives.

3.3.1 Formal Analysis of MMO

Compared with the original space of f_t and f_a, MMO essentially does two main geometric operations to transform the original space of the 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; (2) it rotates the scaled configurations by $45°$ clockwise and then dilates them on both f_t and f_a by a factor of $\sqrt{2}$. Geometrically, MMO in Equation 3 can be decomposed via the following transformation metrics in linear algebra:

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

$$= \overbrace{\begin{bmatrix} f_t(\boldsymbol{x}) + w f_a(\boldsymbol{x}) \\ f_t(\boldsymbol{x}) - w f_a(\boldsymbol{x}) \end{bmatrix}}^{\text{MMO space}}$$

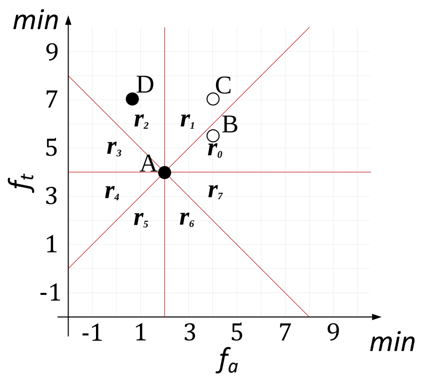
5. In our FSE work [27], we found that different forms of the auxiliary performance objectives (e.g., linear and quadratic) do not lead to significantly different results, hence in this work, we use the linear form, which is the simplest version of the MMO model.

whereby $\sin\frac{\pi}{4} = \frac{\sqrt{2}}{2}$ and $\cos\frac{\pi}{4} = \frac{\sqrt{2}}{2}$, hence the rotation angle is $\frac{\pi}{4}$ (i.e., $45°$ clockwise) and both axes are dilated by a factor of $\sqrt{2}$ thereafter to create a rotation matrix of 1 and -1.

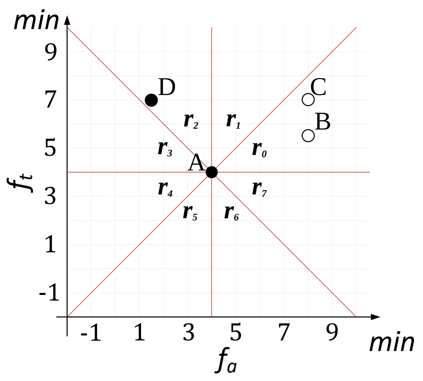
To better understand how the transformation works in MMO, suppose that there are four configurations $\boldsymbol{A}$, $\boldsymbol{B}$, $\boldsymbol{C}$, and $\boldsymbol{D}$, where $\boldsymbol{A}$ is one of the nondominated configurations, as shown in Figure 3a. The areas around $\boldsymbol{A}$ can be divided into eight regions every $45°$, starting counterclockwise from the region where $\boldsymbol{B}$ is located and they are marked as $r_0, r_1, ..., r_7$, respectively. Note that since f_a and f_t are to be minimized in the original space of PMO (the same for g_1 and g_2 in the MMO space), the configurations that are dominated by $\boldsymbol{A}$ will be those in its first quadrant, i.e., in regions r_0 and r_1. Therefore, we can precisely define the dominance relations between configurations and $\boldsymbol{A}$ with respect to the regions of $\boldsymbol{A}$ via the following:

- Configurations in r_0 and r_1 of $\boldsymbol{A}$ (including the boundaries) will be dominated by $\boldsymbol{A}$.
- Likewise, configurations in r_4 and r_5 of $\boldsymbol{A}$ (including the boundaries) will dominate $\boldsymbol{A}$ (not applicable if $\boldsymbol{A}$ is Pareto optimal).
- Finally, configurations in r_2, r_3, r_6, and r_7 will be nondominated to $\boldsymbol{A}$, including the boundaries except those adjacent to r_0, r_1, r_4, and r_5.

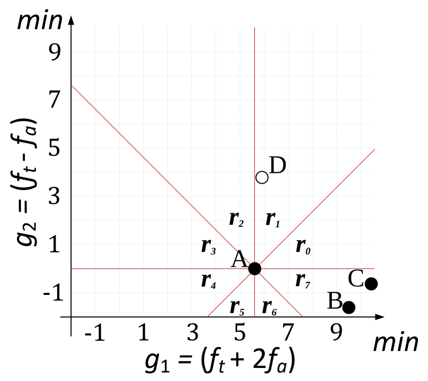
Following the rotation in MMO, the configurations in the regions with respect to $\boldsymbol{A}$ will also be rotated with $\boldsymbol{A}$, and hence they would be in different regions compared with where they were before, which might cause shifts in their relative dominance relationship to $\boldsymbol{A}$—the key that makes MMO works effectively in tuning software configuration. Assuming that $w = 1$, i.e., no scaling (Figure 3b) and hence we can focus on discussing the impact of rotation in MMO, from Figure 3c it is not difficult to see that, compared to $\boldsymbol{A}$, all the configurations will be moved clockwise to their adjacent regions after the rotation of $45°$. Using the case of $\boldsymbol{A}$ as an example again, configuration $\boldsymbol{D}$ in r_2 will be moved to r_1; $\boldsymbol{C}$ in r_1 will be moved to r_0; $\boldsymbol{B}$ in r_0 will be moved to r_7. As such, we can generalize the following rule:



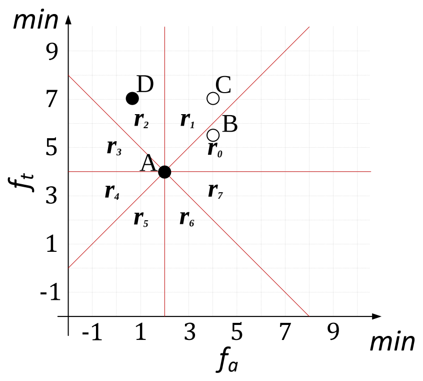
(a) Original space



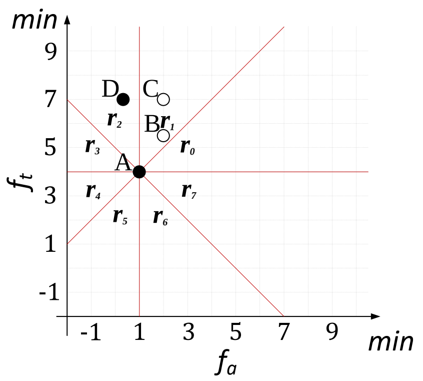
(b) Scaling at $w = 2$ in MMO



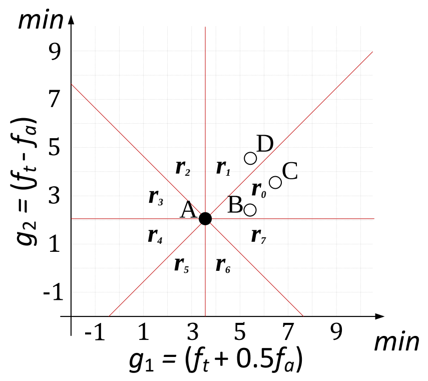
(c) Rotation/dilation at $w = 2$ in MMO



(d) Original space



(e) Scaling at $w = 0.5$ in MMO



(f) Rotation/dilation at $w = 0.5$ in MMO

Fig. 4: An illustration of the impact of w in MMO. The formats are the same as in Figure 3. (a), (b), and (c) delineate the effect of an increased weight, i.e., $w = 2$; (d), (e), and (f) explain the effect of a decreased weight, i.e., $w = 0.5$.

MMO Rule

A configuration in a region r_i will be in a new region r_j following the rotation in MMO, and they satisfy the condition below:

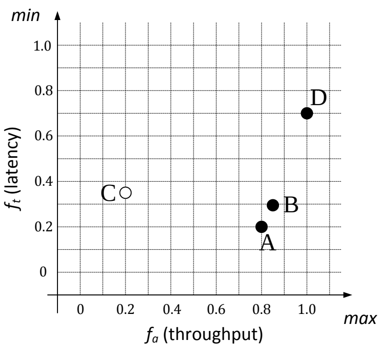
$$j = (i + 7) \bmod 8 \tag{5}$$

Finally, f_a and f_t of the configurations are dilated by $\sqrt{2}$ times for normalizing the coefficients of the rotation matrix to 1, and hence such dilatation has no effect on the dominance relationships between configurations.

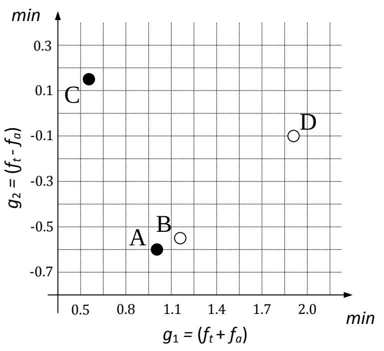
From the above, it is intuitive to understand how the MMO can change the dominance relationships in the space: by moving the configurations between the regions with respect to a particular configuration, it is possible to make them change from dominated (comparable) to nondominated (incomparable), e.g., from r_0 to r_7; or vice versa, e.g., from r_2 to r_1, thereby altering the number of incomparable configurations (in the sense of Pareto dominance) during the tuning. Such an amendment of the Pareto dominance relationships determines the effectiveness of MMO, as either too many or too few incomparable configurations throughout the tuning would be harmful since the former causes a loss of search direction while the latter increases the possibility of being trapped at local optima.

Once it is clear how rotation affects the configurations in the MMO space, we can now explain the role of w in the MMO model. Since the value of weight w determines the factor of scaling, it is not hard to imagine that increasing w stretches all configurations horizontally on f_a; conversely, decreasing w shrinks the configurations horizontally over f_a. Figures 4a, 4b, and 4c give a concrete example using the same configurations as before. When changing from $w = 1$ to $w = 2$, the f_a for all configurations are stretched by 2 times, causing configuration $\boldsymbol{C}$ to move from region r_1 to r_0 before the rotation (Figure 4b). This means that the relative positions of configurations $\boldsymbol{A}$, $\boldsymbol{B}$, $\boldsymbol{C}$, and $\boldsymbol{D}$ will rotate from Figure 4b to Figure 4c. As such, $\boldsymbol{C}$, which should be dominated by $\boldsymbol{A}$ when rotating under $w = 1$, now will become nondominated to $\boldsymbol{A}$ after the rotation. In contrast, if we change $w = 1$ to $w = 0.5$ (Figures 4d, 4e, and 4f), the configurations will be shrunk on f_a by 0.5 times, where configuration $\boldsymbol{B}$ will be moved from regions r_0 to r_1 before rotation (Figure 4e). As a result, the relative positions of configurations $\boldsymbol{A}$, $\boldsymbol{B}$, $\boldsymbol{C}$, and $\boldsymbol{D}$ will rotate from Figure 4e to Figure 4f. In this case, $\boldsymbol{B}$, which should be nondominated to $\boldsymbol{A}$ when rotating under $w = 1$, now will be dominated by $\boldsymbol{A}$ after the rotation.

The above indicates a simple rule to understand the role of w in MMO: a larger w suggests a bigger stretch on f_a, making more of the configurations become incomparable following the rotation, 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, making the two meta-objectives linearly conflicted, hence all configurations



(a) The original target-auxiliary space (i.e. the PMO model)



(b) The meta-objective space (i.e. the MMO model)

Fig. 5: An illustration of comparison between (a) the PMO model and (b) the MMO model on STORM, where the target performance objective is latency (to minimize) and the auxiliary performance objective is throughput (to maximize). Both of them are normalized and the weight is 1.0 in the MMO model. Let us say $\boldsymbol{A}$, $\boldsymbol{B}$, $\boldsymbol{C}$ and $\boldsymbol{D}$ be a set of four configurations to be considered. Out of them, one needs to select two (e.g., in order to put some better configurations into the next-generation population in a multi-objective optimizer). The solid circle means the configuration being Pareto optimal to the set, and the hollow one is the dominated configuration.

are incomparable if they differ on f_a and render the tuning with no guidance. On the other hand, A smaller w means a bigger shrink on f_a, thus more configurations become comparable after the rotation, 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, thereby making the tuning more difficult to jump out of the local optima. In general, neither too large nor too small w is ideal; yet, how large (or small) is considered as too large (or too small) is really case-dependent.

While here we use a nondominated configuration $\boldsymbol{A}$ as the example, it is worth noting that the analysis discussed thereof is applicable to any configurations. As such, the dominance relationships between any pair of configurations can be potentially changed by the scaling of w and rotation introduced in MMO. This as a whole would affect the behaviors and focus (exploration vs. exploitation) of the configuration tuning process regardless of the underlying multi-objective optimizer.

3.3.2 The Characteristics of MMO

To better understand the characteristics of the MMO model derived from our analysis and how the aforementioned two goals can be achieved, Figure 5 gives an example of STORM on how it distinguishes between different configurations, in comparison with the PMO model, where we assume that latency is the target performance objective f_t and throughput is the auxiliary performance objective f_a. Suppose that there is a set of four configurations $\boldsymbol{A}$, $\boldsymbol{B}$, $\boldsymbol{C}$ and $\boldsymbol{D}$. Let us say if we want to select two from them based on their fitness (e.g., in order to put some better configurations into the next-generation population in a multi-objective optimizer, such as NSGA-II). For the PMO model (Figure 5a) that minimizes latency and maximizes throughput, the configuration $\boldsymbol{D}$, which performs extremely poor on latency, will certainly be selected by any multi-objective optimizer, since it is Pareto optimal and also less crowded than the other Pareto optimal configuration $\boldsymbol{A}$ and $\boldsymbol{B}$. In contrast, for the MMO model (Figure 5b) which minimizes the two meta objectives, the two configurations that will be selected are $\boldsymbol{A}$ and $\boldsymbol{C}$ (since they are the only two Pareto optimal ones).

It is worth noting that for the single-objective optimization model (which only considers latency), the two chosen configurations will be $\boldsymbol{A}$ and $\boldsymbol{B}$. However, since $\boldsymbol{C}$ and $\boldsymbol{A}$ behave much more differently than $\boldsymbol{B}$ and $\boldsymbol{A}$ on the throughput, it is more likely that they are located in distant regions in the configuration landscape; thus preserving $\boldsymbol{C}$ rather than $\boldsymbol{B}$ (when $\boldsymbol{A}$ is preserved) is generally more likely to help the search to escape from the local optimum.

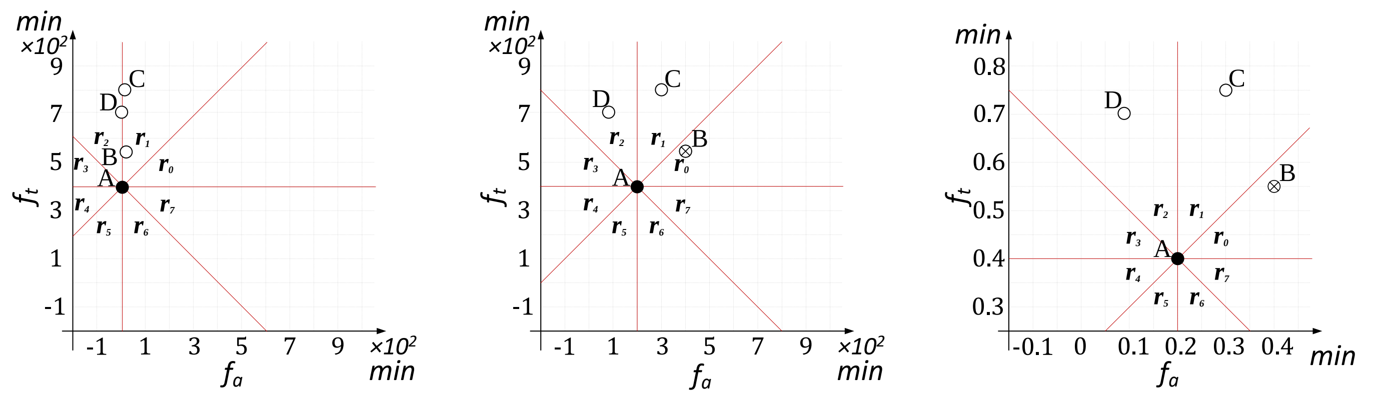
In the following, we provide several remarks to help further grasp the characteristics of the MMO model.

Remark 1. The global optimum of the original single-objective problem (i.e., the configuration with the best target performance objective) is Pareto optimal (e.g., the configuration $\boldsymbol{A}$ in the example of Figure 5). This can be derived immediately by contradiction from Equation (3) or the analysis of rotation from Section 3.3.1.

Remark 2. A similar but more general observation is that a configuration will never be dominated by another that has a worse target performance objective. That is, if configuration $\boldsymbol{x_1}$ has a better target performance objective than $\boldsymbol{x_2}$ (i.e., $f_t(\boldsymbol{x_1}) < f_t(\boldsymbol{x_2})$), then whatever their auxiliary performance objective values are, $\boldsymbol{x_2}$ will not be better than $\boldsymbol{x_1}$ on both g_1 and g_2; in the best case for $\boldsymbol{x_2}$, they are nondominated to each other (e.g., the configuration $\boldsymbol{B}$ versus $\boldsymbol{C}$ in Figure 5). Indeed, according to the analysis from Section 3.3.1, there is no way for $\boldsymbol{x_1}$ to be moved to the region r_0 or r_1 of $\boldsymbol{x_2}$ as $\boldsymbol{x_1}$ can only be in the r_4 to r_7 of $\boldsymbol{x_2}$ before the rotation.

Remark 3. The above two remarks apply to the target performance objective, but not to the auxiliary performance objective. This is a key difference from the PMO model, where both objectives hold these remarks. An example of the consequence is the configuration $\boldsymbol{D}$ of Figure 5, which is meaningless to the original problem, but treated as being optimal in PMO and not in MMO.

Remark 4. Our MMO model does not bias to a higher



(a) Original space where f_a has a much smaller scale than that of f_t

(b) If no normalization is used; scaling at $w = 100$ in MMO

(c) Anticipated effect of normalization; scaling at $w = 1$ in MMO

Fig. 6: An illustration of the impact of the scale discrepancy between performance objectives and the anticipated normalization effect in MMO. $\boldsymbol{A}$, $\boldsymbol{B}$, $\boldsymbol{C}$, and $\boldsymbol{D}$ are four configurations with the auxiliary and target performance objective values as $(2 \times 10^2, 4)$, $(4 \times 10^2, 5.5)$, $(3 \times 10^2, 8)$, respectively. $\otimes$ **and** $\bigcirc$ **denote the configurations that will still be/become nondominated and dominated, respectively, after the rotation in MMO;** $\bullet$ is the reference nondominated configuration.

or lower value on the auxiliary performance objective, in contrast to PMO. Indeed, as in **Property 3**, we do not know for certain what value of the auxiliary performance objective corresponds to the best target performance objective.

Remark 5. Configurations with dissimilar auxiliary performance objective values tend to be incomparable (i.e., nondominated to each other) even if one is fairly inferior to the other on the target performance objective. For example, the configuration $\boldsymbol{C}$ in Figure 5, which has worse latency than $\boldsymbol{A}$, is not dominated by $\boldsymbol{A}$ as their throughput are rather different. In contrast, the configuration $\boldsymbol{B}$, which even has better latency than $\boldsymbol{C}$, is dominated by $\boldsymbol{A}$, as they are similar on throughput. This enables the model to keep exploring diverse promising configurations during the search, thereby a higher chance to find the global optimum.

Remark 6. If two configurations have the same value of auxiliary performance objective f_a, then they are always subject to the dominance relation (i.e., either dominating or being dominated). This is because, if configuration $\boldsymbol{x_1}$ has the same f_a value as $\boldsymbol{x_2}$, then $\boldsymbol{x_1}$ can only be on the boundary between r_1 and r_2 (or r_5 and r_6) for $\boldsymbol{x_2}$, meaning that after the rotation, it can only be on the boundary between r_0 and r_1 (or r_4 and r_5) for $\boldsymbol{x_2}$, in which case their relationships are always dominated (comparable) regardless the f_t value.

Remark 7. If two configurations have the same value of the target performance objective f_t, then they are always nondominated to each other in the MMO model. This is because, if configuration $\boldsymbol{x_1}$ has the same f_t value as $\boldsymbol{x_2}$, then $\boldsymbol{x_1}$ can only be on the boundary between r_0 and r_7 (or r_3 and r_4) for $\boldsymbol{x_2}$, meaning that after the rotation, it can only be on the boundary between r_7 and r_6 (or r_2 and r_3) for $\boldsymbol{x_2}$, in which case their relationships are always nondominated (incomparable) regardless the f_a value and its scaling.

From **Remarks 1–5**, we can see that the MMO model is capable of focusing on optimizing the target performance objective (**Goal 1**) while mitigating the search from being trapped in local optima (**Goal 2**). In the following sections, on the basis of **Remarks 6** and **7**, together with the analysis from Section 3.3.1, we will explain why and how the weight parameter w in the MMO model can be removed by changing the normalization method for the model.

3.4 Normalization for MMO Model in the FSE work [27]

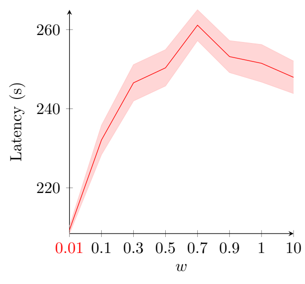
The above analysis assumes an ideal scenario, i.e., the target and auxiliary performance objectives are of similar scale. This is, however, unrealistic for practically tuning configuration. For example, measuring the difference between latency results often reaches the magnitude of 5 order while CPU load merely differs at the scale of a few percentages. The consequence is that the appropriate w value, which enables a good balance between exploitation and exploitation for MMO, can be either very large or very small depending on the case, leading to high difficulty in setting the w.

Figure 6 gives an example. As can be seen, from Figure 6a, f_t has roughly $100\times$ greater scale than that of f_a, and thereby from the coordinates, the configurations will be shrunk along f_a to the boundary between r_1 and r_2 for $\boldsymbol{A}$. This means that, after the rotation in MMO, $\boldsymbol{A}$ will dominate all other configurations and make them comparable, which is harmful. To mitigate such, one would need to give a rather large w value, i.e., $w = 100$, that enables a more reasonable incomparability among the configurations, i.e., some are nondominated while some others are dominated by $\boldsymbol{A}$ (Figure 6b). Yet, since we do not normally have a precise understanding as to what extent the scales between different performance objectives differ beforehand, one would need to examine a wide range of possible w values.

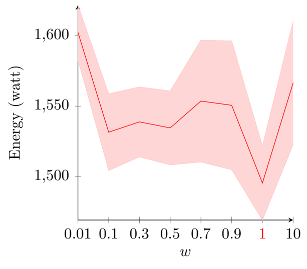
To ease the above, in the FSE work [27], we obtain more commensurable $f_t(\boldsymbol{x})$ and $f_a(\boldsymbol{x})$ via the following normalization (we call it FSE normalization thereafter):

$$f(\boldsymbol{x}) = \frac{f^o(\boldsymbol{x}) - f^o_{lower}}{f^o_{upper} - f^o_{lower}} \tag{6}$$

where $f^o(\boldsymbol{x})$ denotes the original value of the configuration $\boldsymbol{x}$ on the performance objective f, and f^o_{lower} and f^o_{upper} are the global lower and upper bounds on that performance



(a) STORM



(b) x264

Fig. 7: The performance of eight w values for the original MMO model with the FSE normalization on two exampled systems (the best w is highlighted). With 50 repeated runs under 600 and 2500 measurement budgets respectively, STORM consumes a total of $600 \times 8 \times 50 = 2.4 \times 10^5$ measurements while x264 needs $2500 \times 8 \times 50 = 10^6$ measurements. Suppose that each measurement takes one second, it would need around 2.7 and 11.6 days merely to identify the best w setting.

objective for the software, respectively. That is, the true scale of the performance objective is used as the bounds.

In practical software configuration tuning, however, f^o_{lower} and f^o_{upper} are likely to be unknown a priori. Therefore in our FSE work, these bounds are updated by using the maximum and minimum values discovered so far during the tuning to approximate the true scales. Note that using the true scales of the objectives (if known) or their close approximations for normalization is a widely used method in SBSE [98], [80], [89], [32], [4].

Essentially, normalization plays a similar role to w in that they both scale the relative positions of the configurations (but the w primarily works on f_a). As such, with the FSE normalization, our hope was that its resulting scaling could reduce the range of the ideal w values, hence relieving the effort of adjusting it. In the best scenario, we anticipate that the configurations on f_t and f_a can be naturally scaled to ideal positions even when $w = 1$ (i.e., no scaling), thereby there is a good mix of incomparable and comparable configurations after the rotation, leading to more balanced exploitation and exploration, e.g., in Figure 6c.

3.5 What was Wrong?

Indeed, we have shown that the FSE normalization method can be effective in narrowing down the ideal range of w to achieve superior results [27], but with one ineffective outcome: our analysis thereafter reveals that the weight w in the MMO model remains a highly sensitive parameter, even within a narrower range, and finding the right setting for a system still requires much effort of trial and error. In [27] and this work (Section 5), we examined a set of the weight settings for MMO model (i.e., $0.01, 0.1, 0.3, 0.5, 0.7, 0.9, 1.0, 10$)[6]. A key finding is that the weight achieving the best performance differs drastically on different configurable software systems: as shown in Figure 7, some systems work better with a tiny weight value, e.g., $w = 0.01$ for the latency of STORM under the WORDCOUNT benchmark, while some others do best with a much bigger value, e.g., $w = 1$ for the energy usage of system x264. In what follows, we will explain what caused this issue that deviates from our original expectation by means of both theoretical analysis and empirical evidence.

6. We chose these values because they are originally used in the FSE work and it is found that w values outside $[0.01, 10]$ only degrade the results.

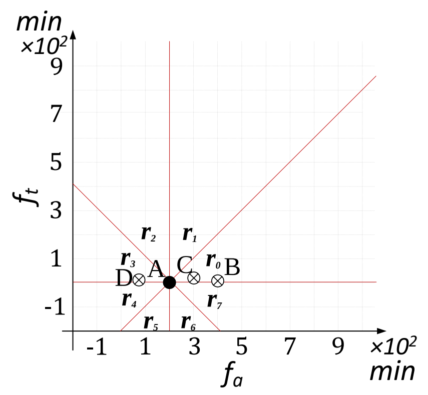
3.5.1 An Analysis

The above occurrence is due to the severe discrepancy between the range of the current search population and the performance objectives' scales in software configuration tuning, which obscures the benefit of the normalization schema we used for the FSE work. To provide a sound analysis thereof, recall the analysis on the effect of w and the rotation from Section 3.3.1, using lower/upper bound in the normalization might lead to two cases in the presence of discrepant performance objective values:

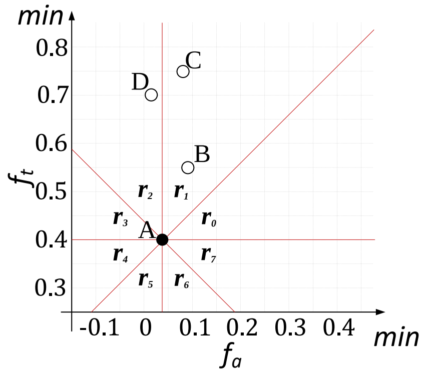
Case 1: If the f_a of the configurations in the population shrinks into a tiny range compared to its true objective scale in the whole search space (while the f_t does not), then the f_a in the population after the normalization (Equation 6) will be very close (see **Remark 6**). Figure 8a shows an example, from which we see that without normalization, all configurations will be nondominated after rotation, i.e., the scale of f_t (e.g., CPU load) is much smaller than that of f_a (e.g., latency), which is not ideal. However, when the bounds for f_a is $[0, 5000]$ while that for f_t is $[0, 10]$, with the FSE normalization (Figure 8b), all configurations (except $\boldsymbol{A}$) will be shrunk towards the boundary between r_1 and r_2 of $\boldsymbol{A}$ (or r_5 and r_6), meaning that they will become dominated (or dominate to) by $\boldsymbol{A}$ after the rotation. This is also devastating to the tuning since too many comparable configurations will over-emphasize exploitation. To mitigate such, we need to set a larger w for stretching f_a, i.e., in this case, $w = 5$ as shown in Figure 8c, thereby the number of incomparable configurations after rotation can be more reasonable to balance the exploration and exploitation.

Case 2: On the other hand, if the f_t in the population evolves into a range that is tiny compared to its objective scale in the whole search space (while the f_a does not), then the values of the f_t of configurations in the current population after the normalization (Equation 6) will be very close (see **Remark 7**). As can be seen in Figure 8d, suppose that before normalization the scale of f_a (e.g., CPU load) is much smaller than that of f_t (e.g., latency), then this will cause $\boldsymbol{A}$ to dominate all other configurations when rotating, which is harmful. With the FSE normalization under the bounds for is $[0, 5000]$ for f_t and $[0, 10]$ for f_a, as in Figure 8e, all configurations (except $\boldsymbol{A}$) will be shrunk towards the boundary between r_3 and r_4 (or r_0 and r_7) of $\boldsymbol{A}$, causing more configurations to become nondominated to $\boldsymbol{A}$ after the rotation, hence creating many incomparable configurations that focus too much on exploration that would also harm the guidance of tuning. Likewise, to relieve such a case, we need to set a smaller w for shrinking f_a, e.g., $w = 0.2$ as shown in Figure 8f, thereby the number of incomparable configurations after rotation is more appropriate, enabling the tuning to favor towards exploitation that reaches a balance.

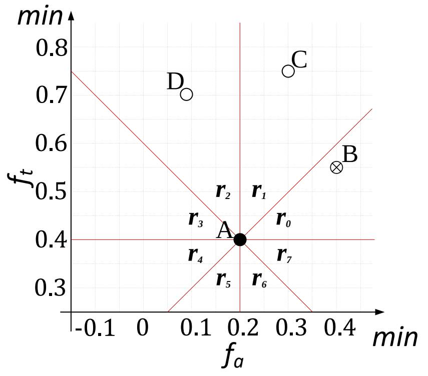
As a result, from the above, it is clear that although the FSE normalization helps to reduce the ideal ranges of w values (0.2 and 5 instead of the range on, e.g., $[0.001, 1000]$), it can still negatively influence the appropriate number of



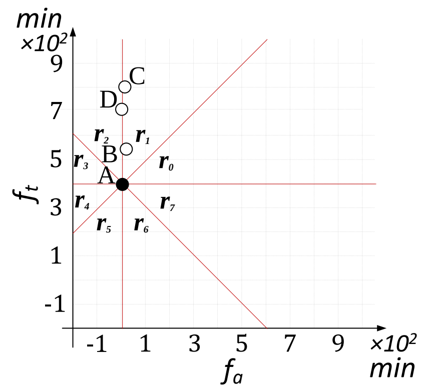
(a) Original space where f_t has a much smaller scale than that of f_a



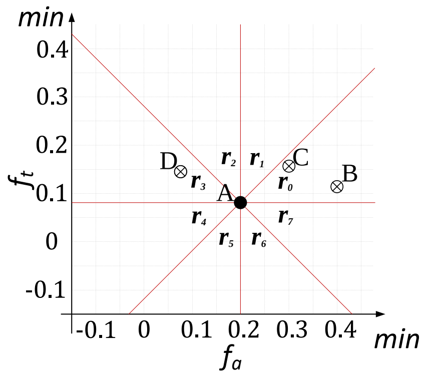
(b) FSE normalization at $f_a \in [0, 5000]$ / $f_t \in [0, 10]$; scaling at $w = 1$ in MMO



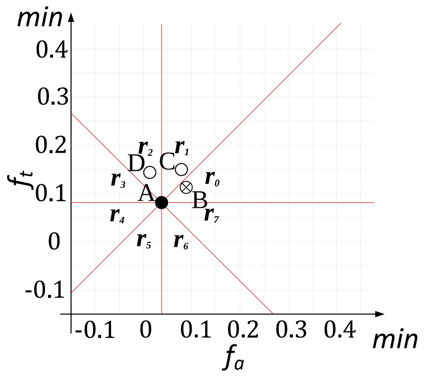
(c) FSE normalization at $f_a \in [0, 5000]$ / $f_t \in [0, 10]$; scaling at $w = 5$ in MMO



(d) Original space where f_a has a much smaller scale than that of f_t

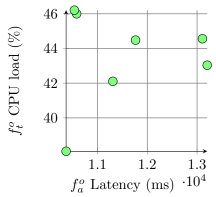


(e) FSE normalization at $f_a \in [0, 10]$ / $f_t \in [0, 5000]$; scaling at $w = 1$ in MMO

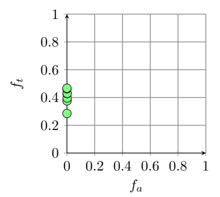


(f) FSE normalization at $f_a \in [0, 10]$ / $f_t \in [0, 5000]$; scaling at $w = 0.2$ in MMO

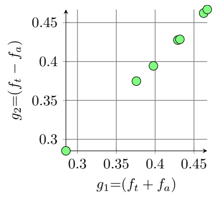
Fig. 8: Illustration of why the MMO designs in our FSE work is highly sensitive to w. $\boldsymbol{A}$, $\boldsymbol{B}$, $\boldsymbol{C}$, and $\boldsymbol{D}$ are four configurations with the auxiliary and target performance objective values as $(2, 4 \times 10^2)$, $(4, 5.5 \times 10^2)$, $(3, 8 \times 10^2)$, and $(0.75, 7 \times 10^2)$ in (a), respectively. In (d), the values are $(2 \times 10^2, 4)$, $(4 \times 10^2, 5.5)$, $(3 \times 10^2, 8)$, and $(0.75 \times 10^2, 7)$, respectively. The format is the same as Figure 6. (a), (b), and (c) show the case where the f_a is normalized into a tiny range with bounds of $[0, 5000]$ as opposed to that of $[0, 10]$ for f_t; (d), (c), and (e) demonstrate the case where the f_t is normalized into a tiny range with bounds of $[0, 5000]$ as opposed to that of $[0, 10]$ for f_a.



(a) Original

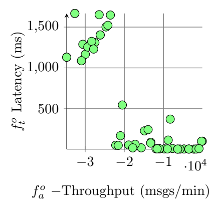


(b) Normalized

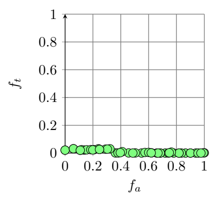


(c) In the MMO space

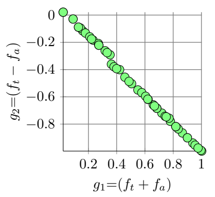
Fig. 9: An intermediate population of configurations for system LRZIP generated by our MMO model with the FSE normalization [27] on top of NSGA-II. Scales in the original population are adjusted for visibility.



(a) Original



(b) Normalized



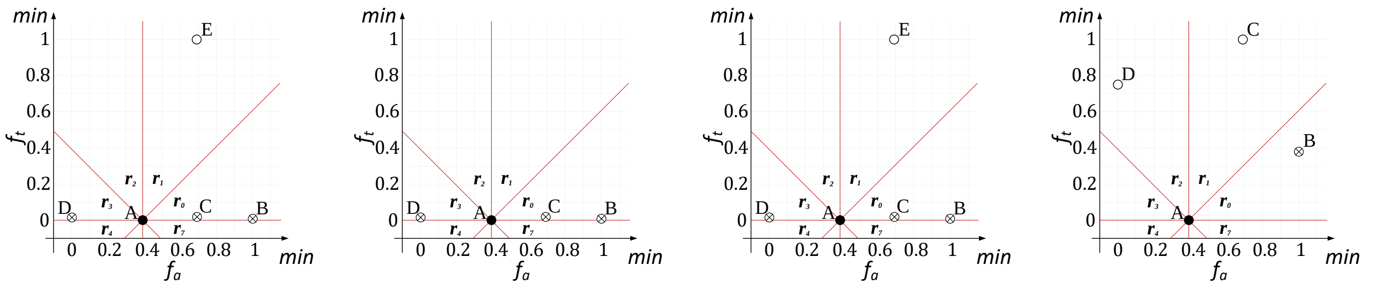
(c) In the MMO space

Fig. 10: An intermediate population of configurations for system STORM/WC generated by our MMO model with the FSE normalization [27] on top of NSGA-II. Scales in the original population are adjusted for visibility.

incomparable configurations following the rotation in MMO, because a very large upper bound for one performance objective might be reached and being used throughout the tuning, even if such an extreme configuration has been ruled out later on. Therefore, we still need to non-trivially adjust w to mitigate such a side-effect from the FSE normalization, which collectively influences the effects of rotation. This is the key reason that the MMO designs in our FSE work remain highly sensitive to its only parameter w.

3.5.2 Some Empirical Evidences

To demonstrate the devastating impact of FSE normalization on the MMO model, Figure 9 illustrates an intermediate population of MMO model during the tuning for system LRZIP. As can be seen from Figure 9b, since the range of auxiliary performance objective in the population becomes



(a) FSE normalization in MMO at an iteration where $\boldsymbol{E}$ is newly discovered

(b) FSE normalization in MMO after some iterations when $\boldsymbol{E}$ is eliminated

(c) New normalization in MMO at an iteration where $\boldsymbol{E}$ is newly discovered

(d) New normalization in MMO after some iterations when $\boldsymbol{E}$ is eliminated

Fig. 11: A simple illustration of why the new normalization is more useful. $\boldsymbol{A}$, $\boldsymbol{B}$, $\boldsymbol{C}$, $\boldsymbol{D}$ and $\boldsymbol{E}$ are five configurations with the auxiliary and target performance objective values as $(2, 4)$, $(4, 5.5)$, $(3, 8)$, $(0.75, 7)$, and $(3, 5 \times 10^2)$, respectively. The format is the same as Figure 6. (a) and (b) show the case of MMO under the FSE normalization; (c) and (d) demonstrate the case of MMO under the new normalization. All cases have no scaling in MMO, i.e., $w = 1$.

"very small" (around $[1.03 \times 10^4 ms, 1.3 \times 10^4 ms]$), compared to the objective scale ($[10^4 ms, 1.7 \times 10^6 ms]$, after the normalization the auxiliary performance objective's values become tiny, condensing in the range of $[0, 1.7 \times 10^{-3}]$ only. This, as we discussed during the analysis, can lead to all the configurations in the population being either dominating or dominated by each other within the transformed space of our MMO model (Figure 9c), which effectively means that the problem degenerates to the original single-objective problem, where the configurations are discriminative virtually based on their target performance objective (thus easily being trapped in local optima).

Figure 10 gives yet another example, where we visualize an intermediate population of MMO model (with the FSE normalization) during the tuning for system STORM/WC. As can be seen in Figure 10b, since the range of the target performance objective (latency) in the population becomes "very small" (around $[95ms, 1800ms]$), compared to the objective scale ($[3ms, 55209ms]$), after the normalization the values of the target performance objective become tiny, condensing in the range of $[0, 0.03]$ only. The auxiliary performance objective (throughput), in contrast, are more evenly spread over the range of $[0, 1]$ after the normalization. As with our analysis, this can lead to all the configurations in the population being nondominated to each other within the transformed MMO space (Figure 10c). Unfortunately, all configurations in the population being nondominated is detrimental to the search since there is no selection pressure (i.e., discriminative power); everyone is incomparable even the one with the best target performance objective.

3.6 A New Normalization

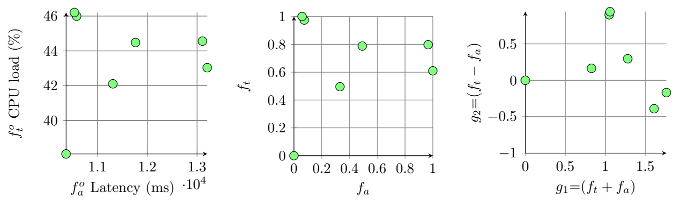
The above analysis and observations suggest that the FSE normalization based on the (approximate) true scales of the performance objectives may not be suitable for the MMO model. Fortunately, this can be fixed by considering the current population as the basis of bounds in the normalization, which is the key extension in this work. That is, we replace Equation 6 with the following:

$$f(\boldsymbol{x}) = \frac{f^o(\boldsymbol{x}) - f^o_{min}}{f^o_{max} - f^o_{min}} \tag{7}$$

where $f^o(\boldsymbol{x})$ denotes the original value of the configuration $\boldsymbol{x}$ on the performance objective f, and f^o_{min} and f^o_{max} are the maximum and minimum values of the current population on f, respectively. As such, instead of using the global bounds throughout the search, the local bounds (in the population of configurations of every generation along with the evolution) are used in the normalization.

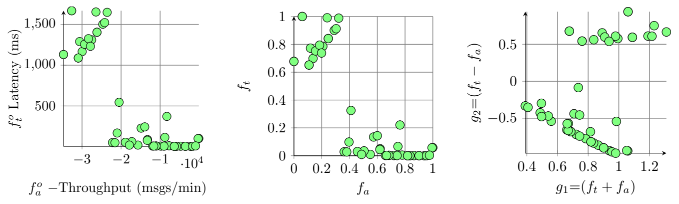
To better explain how the new normalization differs from the FSE normalization in the MMO space, Figure 11 shows an example. Here, from Figure 11a and 11c, suppose that during the tuning a configuration $\boldsymbol{E}$ with a very large value of f_t is discovered, then this will cause both normalizations to shrink the configurations along f_t, leaving a negative impact as most configurations will become nondominated (incomparable) after rotation. However, it is possible that $\boldsymbol{E}$ will be subsequently ruled out due to it being the only configuration dominated by $\boldsymbol{A}$ in the MMO space. Yet, with the FSE normalization (Figure 11b), the bounds remain unchanged hence all configurations are still nondominated, i.e., the side effect left by $\boldsymbol{E}$ will remain present. In contrast, with the new normalization in this work (Figure 11d), the bounds are updated locally within the population of preserved configurations and hence they can be scaled more reasonably, leading to a better mix of comparable and incomparable configurations after $\boldsymbol{E}$ is eliminated. This will strike a good balance between imposing the selection pressure toward the best target performance objective and preserving the diversity of the auxiliary performance objective.

Figures 12 and 13 give the results of the examples from Figures 9 and 10 after the new normalization is implemented, respectively. As can be seen, the configurations in the population after the normalization do not concentrate into one value on either objective (Figures 13b and 12b), and in our MMO space there now exist both dominated and nondominated configurations in the population (Figures 13c and 12c). In this case, there is less need to adjust the w in the MMO model to mitigate the side-effect of normalization, which balances the number of incomparable configurations after rotation, since the two performance objectives after the normalization are always commensurable. As such, we can generally remove the weight, i.e., setting $w = 1$ for all cases.



(a) Original (b) Normalized (c) In the MMO space

Fig. 12: The same population of Figure 9 under the MMO model with the new normalization method in this work. Scales in the original population are adjusted for visibility.



(a) Original (b) Normalized (c) In the MMO space

Fig. 13: The same population of Figure 10 under the MMO model with the new normalization method in this work. Scales in the original population are adjusted for visibility.

3.7 Integrating with an Optimizer

Since MMO model is an optimization model, it can fit with different population-based multi-objective optimizers such as NSGA-II. A pseudo-code for using the MMO model with the normalization on top of NSGA-II has been demonstrated in Algorithm 1. As can be seen, there are two amendments required (the red crossed statements are the code for the FSE work and the green ones are the code changed in this work):

1) Keeping track of the bounds on $f_t(\boldsymbol{x})$ and $f_a(\boldsymbol{x})$ for normalizing both the target and auxiliary performance objectives (lines 6–7 and 23–24). The definitions of those bounds differ depending on the normalization methods, i.e., with lines 23–24 instead of lines 21–22, the bounds are locally restricted to the current population or otherwise, they would be the global bounds so far.
2) Performing the normal Pareto search procedure in NSGA-II within the transformed meta-objective space ($g_1(\boldsymbol{x})$ and $g_2(\boldsymbol{x})$) of MMO model without a weight, instead of the original target-auxiliary space ($f_t(\boldsymbol{x})$ and $f_a(\boldsymbol{x})$), as shown at lines 10, 15, 28, and 30.

Indeed, we do not need to make a significant amount of refactoring on MMO at the code level, but as we will show later on (Section 5), such a simple change can lead to dramatic improvements in its effectiveness while saving the overhead of adjusting the weight. It is worth noting that, proposing a simple method that leads to large improvements is not easy, as this requires in-depth understanding and reasoning about the principles/causes behind the observations, as what we have shown in our theoretical and empirical analysis, which requires a large amount of effort.

4 Experimental Evaluation

In this section, we articulate the experimental methodology for evaluating our MMO model with the new normalization.

Algorithm 1: MMOonNSGA-II

```
Input: Configuration space $\mathcal{V}$; the system $\mathcal{F}$; ~~weight $w$~~
Output: $s_{best}$ the best configuration on $f_t(\boldsymbol{x})$
Declare: bound vectors $\overline{\mathbf{z}}_{\mathbf{max}}$ and $\overline{\mathbf{z}}_{\mathbf{min}}$
1  Randomly initialize a population of $n$ configurations $\mathcal{P}$
2  $\overline{\mathbf{z}}_{\mathbf{max}} = \emptyset$; $\overline{\mathbf{z}}_{\mathbf{min}} = \emptyset$
3  /* measuring $f_t$ and $f_a$ of the configurations in $\mathcal{P}$ on the system */
4  MEASURE($\mathcal{P},\mathcal{F}$)
5  /* initializing the bounds for normalization */
6  $\overline{\mathbf{z}}_{\mathbf{max}}$ = UPDATEUPPERBOUNDS($\mathcal{P}$)
7  $\overline{\mathbf{z}}_{\mathbf{min}}$ = UPDATELOWERRBOUNDS($\mathcal{P}$)
8  /* updating $g_1$ and $g_2$ */
9  ~~COMPUTEMMOMODEL($w,\mathcal{P},\overline{\mathbf{z}}_{\mathbf{max}},\overline{\mathbf{z}}_{\mathbf{min}}$)~~
10 COMPUTEMMOMODEL($\mathcal{P},\overline{\mathbf{z}}_{\mathbf{max}},\overline{\mathbf{z}}_{\mathbf{min}}$)
11 while The search budget is not exhausted do
12     $\mathcal{P}' = \emptyset$
13     while $\mathcal{P}' < n$ do
14         /* selecting parents based on $g_1$ and $g_2$ */
15         $\{s_x, s_y\}$ ←MATING($\mathcal{P}$)
16         $\{o_x, o_y\}$ ←DOCROSSOVERANDMUTATION($\mathcal{V}, s_x, s_y$)
17         /* measuring $f_t$ and $f_a$ for configurations $o_x$ and $o_y$ on the system (if unique) */
18         MEASURE($o_x, o_y, \mathcal{F}$)
19         $\mathcal{P}' \leftarrow \mathcal{P}' \bigcup \{o_x, o_y\}$
20     /* the bounds are reset based on the current population at each generation regardless of the previous bound values */
21     ~~$\overline{\mathbf{z}}_{\mathbf{max}}$ = UPDATEUPPERBOUNDS($\mathcal{P},\overline{\mathbf{z}}_{\mathbf{max}}$)~~
22     ~~$\overline{\mathbf{z}}_{\mathbf{min}}$ = UPDATELOWERRBOUNDS($\mathcal{P},\overline{\mathbf{z}}_{\mathbf{min}}$)~~
23     $\overline{\mathbf{z}}_{\mathbf{max}}$ = UPDATEUPPERBOUNDS($\mathcal{P}$)
24     $\overline{\mathbf{z}}_{\mathbf{min}}$ = UPDATELOWERRBOUNDS($\mathcal{P}$)
25     $\mathcal{U}' \leftarrow \mathcal{P} \bigcup \mathcal{P}'$
26     /* updating $g_1$ and $g_2$ */
27     ~~COMPUTEMMOMODEL($w,\mathcal{U}',\overline{\mathbf{z}}_{\mathbf{max}},\overline{\mathbf{z}}_{\mathbf{min}}$)~~
28     COMPUTEMMOMODEL($\mathcal{U}',\overline{\mathbf{z}}_{\mathbf{max}},\overline{\mathbf{z}}_{\mathbf{min}}$)
29     /* sorting based on $g_1$ and $g_2$ */
30     $\mathcal{U}$ ←NONDOMINATEDSORTING($\mathcal{U}'$)
31     $\mathcal{P}$ ←top $n$ configurations from $\mathcal{U}$
32 return $s_{best}$ ←BESTCONFIGURATION($\mathcal{P}$)
```

To better distinguish this work and the FSE work [27], we use the following terminology:

- **MMO-FSE:** This refers to our MMO model with the normalization method from the FSE work.
- **MMO:** This refers to our MMO model with the new normalization proposed in this work.

All optimization models and optimizers are implemented in Java, using jMetal [35] and Opt4J [65].

4.1 Research Questions

Our experiment answers a few research questions (RQs):

— **RQ1:** How effective is the MMO?

As the most fundamental question, we ask **RQ1** to verify whether our MMO can better help to mitigate the issue of local optima, i.e., by providing better results than the MMO-FSE, PMO, and state-of-the-art single-objective counterparts. However, even if the MMO can lead to promising results by mitigating local optima, it would be less useful if it requires a significantly large amount of resources to do so. Under the same settings as **RQ1**, our second research question is, therefore:

— **RQ2:** How resource-efficient is the MMO?

TABLE 1: Configurable software systems studied.

Software System	Domain	Performance Objectives		# Options	Search Space	Used By
MARIADB	SQL database	O1: latency	O2: CPU load	10	864	[75]
STORM/WC	stream processing	O1: throughput	O2: latency	6	2,880	[70], [51], [75]
VP9	video encoding	O1: latency	O2: CPU load	12	3,008	[75]
STORM/RS	stream processing	O1: throughput	O2: latency	6	3,839	[70], [51], [75]
LRZIP	file compression	O1: latency	O2: CPU load	12	5,184	[75]
MONGODB	no-SQL database	O1: latency	O2: CPU load	15	6,840	[75]
KERAS-DNN/SA	deep learning	O1: AUC	O2: inference time	12	16,384	[67], [52]
KERAS-DNN/ADIAC	deep learning	O1: AUC	O2: inference time	12	24,576	[67], [52]
X264	video encoding	O1: PSNR	O2: energy usage	17	53,662	[70], [82], [75]
LLVM	compiler	O1: latency	O2: CPU load	16	65,436	[70], [75]
TRIMESH	triangle mesh	O1: # iteration	O2: latency	13	239,260	[70], [82]

In **RQ2**, we are interested in examining whether the MMO can utilize the resource (the number of measurements) efficiently when reaching a certain level of performance.

One of the key novelties for MMO, compared with MMO-FSE, is weight-free. Yet, this would be meaningless if the MMO-FSE achieves similarly promising results over different weights on the systems studied; or if the effort for finding the best weight is trivial. Hence, our next **RQ** is:

— **RQ3:** How meaningful is the weight-free design in MMO?

RQ3 seeks to understand two aspects: (1) how does MMO perform when compared with MMO-FSE under different weights; and (2) How much extra resource is required to tune the MMO-FSE for finding a promising weight value.

To reduce unnecessary noise, we investigate **RQ1-3** by directly measuring the systems, which belongs to the measurement-based tuning methods for software configuration tuning [71], [102], [95]. However, there exist studies leveraging on the model-based tuning methods where a surrogate is built to serve as a cheap evaluator to predict the performance of a configuration, under the assumption of the single-objective model. Since the key difference between measurement-based and model-based tuning methods lies in whether a surrogate is used to guide the search, the MMO, which itself is an optimization model, can be considered complementary to the model-based alternative. Therefore, our final research question is concerned with:

— **RQ4:** Can MMO consolidate the existing model-based tuning method?

To that end, we extend FLASH [70] and BOCA [17]—two recent tools from the Software Engineering community for configuration tuning—with our MMO and examine whether its performance can be improved.

4.2 Software Systems

To improve the generality of this work, we chose systems from existing studies according to the following criteria:

- To ensure complexity, we exclude simple systems, i.e., those with less than 10 configuration options and all of them are binary.
- The system should involve at least two performance objectives.
- To expedite the experiments, the system should contain readily available data of the measurements on all the valid configurations.
- To improve the diversity of the subject, for the system under different benchmarks, we use the one with the largest search space and the one with the highest deviation on the performance, providing that the above points are satisfied.

As shown in Table 1, we experiment on 11 real-world software systems and environments that have been commonly used in prior work [70], [51], [67], [52], [75]. They come from diverse domains, e.g., SQL database, video encoding, and stream processing, while having different performance attributes, scale, and search space of valid configurations. Each software system has two performance objectives, which are chosen from prior work [70], [51], [67], [52], [75]. In all experiments, we use each of their two performance attributes as the target performance objective in turn while the other serves as the auxiliary performance objective, leading to 22 cases in total. We apply the same configuration options and their ranges as studied previously since those have been shown to be the key ones for the software systems under the related environment.

Noteworthily, it can be rather expensive even for a single measurement under those systems, e.g., it may take up to 341 seconds to measure a configuration on MONGODB. To ensure realism and expedite the experiments, we use the datasets of those systems collected by existing work, in which each measurement is extracted from 3-5 repeats [51], [75].

4.3 Settings for RQ1, RQ2, and RQ3

4.3.1 Optimizers

For the single-objective optimization model, we examine four state-of-the-art optimizers that are widely used in software configuration tuning, all of which deal with local optima in different ways:

- Random Search (RS) with a high neighbourhood radius to escape from the local optima, as used in [8], [99], [71].
- Stochastic Hill Climbing with restart (SHC-r), which is exploited by [95], [62], aiming to avoid local optima by using different starting points.
- Single-Objective Genetic Algorithm (SOGA) from [7], [80], [78], [84] that seeks to escape local optima by using variation operators.
- Simulated Annealing (SA) that tackles local optima by stochastically accepting inferior configurations as used in [37], [42].

While the MMO does not tie to any specific multi-objective optimizer, we use NSGA-II for the MMO, MMO-

TABLE 2: Measurement search budgets and population sizes.

Software	Size	Budget	Software	Size	Budget
MariaDB	20	400	Storm/WC	50	600
VP9	30	700	Storm/RS	50	900
LRZIP	20	400	MongoDB	20	500
Keras-DNN/SA	20	400	Keras-DNN/Adiac	20	400
x264	50	2,500	LLVM	20	600
Trimesh	20	1,000			

FSE, and PMO in this work, because (1) it has been predominately used for software configuration tuning in prior work when multiple performance attributes are of interest [26], [83], [31], [58], [85]; (2) it shares many similarities with the SOGA that we compare in this work. However, it is worth noting that MMO may not be able to work with some multi-objective optimizers designed for SBSE problems where the objectives are not treated equally, such as [73], [46], [45].

4.3.2 Weight Values for MMO-FSE

In our experiments, we evaluate a set of weight values, i.e., $w \in \{0.01, 0.1, 0.3, 0.5, 0.7, 0.9, 1.0, 10\}$, for the MMO-FSE. Those are merely pragmatic settings, but we found that weight beyond this range only degraded the performance, please kindly refer to Section 3.4 for a theoretical explanation. Further, a similar setting is also what has been commonly followed for SBSE work in general [29]. In this way, we aim to examine whether the MMO-FSE can perform as well as MMO under the best weight chosen from a set of diverse weight values (or indeed worse on all of them).

4.3.3 Search Budget

In this work, we use the number of measurements to quantify the search budget and resource consumed, as it is language-/platform-independent and does not suffer from the interference caused by the background processes of the operating system.

Since one of our goals is to examine how badly a model/optimizer can suffer from trapping at undesired local optima when tuning software configuration, it is important to study the result under reasonable convergence, i.e., increasing the search budget is unlikely to change the outcomes. To that end, for every optimizer/model on each system (and its performance objectives), we examine different search budgets from $\{100, 200, ..., X\}$ where X refers to the smaller one between $3,000$ and the size of the search space. The purpose is to set a search budget as the smallest number of measurements for all optimizers/models, such that they all have less than 10% changes of configuration in the population (or no better configuration found when no population is involved) within the last 10% of the successive measurement count[7]. The settings are similar to those used by existing work which were found in a similar way, e.g., Gerasimou *et al.* [38] set no better configuration found for the last 20% of the iteration count as a sign of convergence; Krall *et al.* [56] use 15% as an indicator of convergence. Note that we additionally monitor the percentage of the population change rather than purely whether a better configuration is found, since Harman [44] suggests that a good sign of little realistic chance of further improvements on population-based optimizers is that the population has become homogeneous. To ensure the realism of the setting, we make sure that the actual time taken for exhausting the search budget does not exceed 48 hours for a run overall. The identified search budgets are then used as the termination criterion in our experiments, as shown in Table 2. It is worth noting that the search budget identified remains much smaller than the corresponding search space. For example, it only allows for measuring 0.42% of the configurations for Trimesh.

Since each measurement has considered the noise [51], [75] and only the profiling of systems is expensive in practice, in each run, we cached the measurements of distinct configurations, which can be reused directly when the same configuration appears again during the tuning. As such, only the distinct configurations would consume the budget.

To account for the stochastic nature of the optimizers, we repeat all experiments 50 runs under the search budget.

4.3.4 Other Parameters

For the other key parameters of the optimizers, we apply the binary tournament for mating selection, together with the boundary mutation and uniformed crossover in SOGA and NSGA-II, as used in prior work [26], [80], [31]. The mutation and crossover rates are set to 0.1 and 0.9, respectively, as commonly set in software configuration tuning [26].

What we could not decide easily is the population size for SOGA and NSGA-II. Therefore, for each software system, we additionally examine a set of population sizes, i.e., $\{10, 20, ..., 100\}$, under the search budget identified previously. Similarly, we set the largest population size that can still ensure there are less than 10% changes of configurations at the last 10% of the measurement count. The results are shown in Table 2. In this way, we seek to reach a good balance between convergence (smaller population change) and diversity (larger population size) under a budget.

4.4 Settings for RQ4

4.4.1 Optimizers

For model-based tuning methods, we consider Flash[8] (TSE'20) [70] and BOCA (ICSE'21) [17] in this work, because

- they are recent efforts from the software engineering community to tune software configuration.
- they have been specifically tailored to cater for the key properties of the tuning problem, e.g., high sparsity and expensive measurements.
- their authors have shown that they outperform other more general configuration tuning approaches, e.g., BOCA is better than TPE [17] and Flash is superior to ϵ-PAL [70], as well as better than some older methods for software configuration tuning, e.g., the one by Jamshidi and Casale [70].
- both have been tested on some of the systems studied in this work, e.g., x264 and LLVM.

In a nutshell, Flash was derived from the Sequential Model-Based Optimization (SMBO) paradigm, which is a

7. For SOGA and NSGA-II, the population size is initially fixed to 10, which is the smallest size that we will examine subsequently.

8. Note that when only a single performance objective matters, Flash uses a single-objective model like the RS, SHC-r, SOGA, and SA studied in this work. Hence, we use the single-objective version of Flash.

Algorithm 2: FLASH

```
Input: Configuration space $\mathcal{V}$; the system $\mathcal{F}$
Output: $s_{best}$ the best configuration on $f_t(\boldsymbol{x})$
Declare: vector of surrogates $\mathcal{M}$ (one for each performance
         objective)
1  Randomly initialize a size of $k$ configurations $\mathcal{P}$
2  MEASURE($\mathcal{P}, \mathcal{F}$)
3  /* removing measured configurations */
4  $\mathcal{V} \leftarrow \mathcal{V} - \mathcal{P}$
5  while The search budget is not exhausted do
6      $\mathcal{M}$ = TRAINCARTS($\mathcal{P}$)
7      /* searching an estimated-best
          configuration for measurement */
8      $o$ = FINDBESTCONFIGURATION($\mathcal{V}, \mathcal{M}$)
9      MEASURE($o, \mathcal{F}$)
10     $\mathcal{V} \leftarrow \mathcal{V} - o$
11     $\mathcal{P} \leftarrow \mathcal{P} + o$
12     if $o$ is measured to be better than $s_{best}$ on $f_t(\boldsymbol{x})$ then
13         $s_{best} = o$
14 return $s_{best}$
```

Algorithm 3: BOCA

```
Input: Configuration space $\mathcal{V}$; the system $\mathcal{F}$
Output: $s_{best}$ the best configuration on $f_t(\boldsymbol{x})$
Declare: vector of surrogates $\mathcal{M}$
1  Randomly initialize a size of $k$ configurations $\mathcal{P}$
2  MEASURE($\mathcal{P}, \mathcal{F}$)
3  while The search budget is not exhausted do
4      $\mathcal{M}$ = TRAINRANDOMFOREST($\mathcal{P}$)
5      $\mathcal{I}$ =GETALLSETTINGSONIMPORTANTOPTIONS($\mathcal{M}, K$)
6      for $i \in \mathcal{I}$ do
7          $c$ =DECAY($j$)
8          $\mathcal{I}$ =GETUNIMPORTANTSETTINGS($c$)
9          $\mathcal{P}' \leftarrow$COMBINEDSAMPLES($i, \mathcal{U}$)
10     /* searching an estimated-best
          configuration for measurement */
11     $o$ = FINDBESTCONFIGURATION($\mathcal{P}'$)
12     MEASURE($o, \mathcal{F}$)
13     $\mathcal{P} \leftarrow \mathcal{P} + o$
14     if $o$ is measured to be better than $s_{best}$ on $f_t(\boldsymbol{x})$ then
15         $s_{best} = o$
16 return $s_{best}$
```

generalization of the Bayesian Optimization (BO) [81]. As shown in Algorithm 2, the basic idea is to build a surrogate that learns the correlation between configurations and their values of a performance objective (line 6). Such a surrogate is then used to guide the search to decide which promising configuration to measure next via an acquisition function (line 8), after which the surrogate would be updated by using the newly measured configuration. Like other measurement-based tuning methods, the process terminates when the search budget is exhausted. Yet, unlike the classic BO, FLASH does two major changes for the problem to tune a single performance objective:

1) The surrogate is a CART [14] instead of the classic Gaussian Process in BO [81].
2) The acquisition function no longer considers uncertainty but solely targets the best-predicted performance value.

Note that FLASH originally uses an exhaustive search to find the best-predicted configuration at each sampling iteration (line 8), but this may not be ideal for our study because of two reasons: (1) exhaustively traversing the whole configuration space itself is still a lengthy process, especially on some of the large systems. For example, on each iteration for TRIMESH, it can take several minutes on a standard machine to run even for a surrogate. (2) Since the surrogate is not always accurate [102], the exhaustive search could amplify the side effects caused by the errors in misleading the search. Therefore, we replace the exhaustive search with a random search, which works well and has been recommended as a replacement for SMBO [8].

Similar to FLASH, BOCA (Algorithm 3) also leverages Bayesian Optimization but it uses Random Forest [13] and Expected Improvement [53] as the surrogate and acquisition function, respectively. Further, BOCA takes the top K most important configuration options into account based on the rank from the Random Forest; it then creates a set of candidate configurations that cover c settings for the unimportant options combined with every setting of the important ones, where c is determined proportionally to the search progress (lines 5-9). The one with the best acquisition value from the set is then measured.

4.4.2 Search Budget

To ensure fairness, we set the same search budget as used in the original work of FLASH [70], i.e., 50 measurements. Further, we also use the same initial sample size ($k = 30$ in Algorithm 2) to pre-train the surrogate. As for the search process with the surrogate, we allow for 1,000 surrogate evaluations (including redundant ones) which is a typical setting from the other work for optimizing the surrogate when an exhaustive search is undesirable [54], [76].

Similar to **RQ1-3**, each experiment is repeated 50 runs.

4.5 Statistical Validation

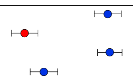
We use the following methods for statistical test:

— **Non-parametric test:** To verify statistical significance, we leverage the Wilcoxon signed-rank test [94] (for paired comparisons between two approaches) and Kruskal-Wallis test [66] (for multiple comparisons). In particular, to understand which pairwise comparisons are significant in the Kruskal-Wallis test, we use Dunn's test [91] as the post-hoc method together with the Holm-Bonferroni correction [2], which will significantly reduce the chances of Type-I error. All of the above are widely used non-parametric test for SBSE and has been recommended in software engineering research for their strong statistical power [6]. The standard $a = 0.05$ is set as the significance level over 50 runs.

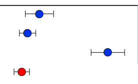
— **Effect size:** To ensure the differences are not generated from a trivial effect, we use $\hat{A}_{12}$ [93] to verify the effect size of the comparisons on target performance objectives over 50 runs. According to Vargha and Delaney [93], when comparing our MMO and its counterpart in this work, $\hat{A}_{12} > 0.5$ denotes that the MMO is better for more than 50% of the times (MMO wins); MOO will lose if $\hat{A}_{12} < 0.5$ and it is a tie when $\hat{A}_{12} = 0.5$. In particular, $0.56 \leq \hat{A}_{12} < 0.64$ indicates a small effect size while $0.64 \leq \hat{A}_{12} < 0.71$ and $\hat{A}_{12} \geq 0.71$ mean a medium and a large effect size, respectively.

As such, we say a result of the comparison is statistically significant only if it has $\hat{A}_{12} \geq 0.56$ (or $\hat{A}_{12} \leq 0.44$) and $p < 0.05$ (after correction if needed).

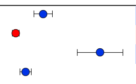
TABLE 3: Comparing MMO with the other state-of-the-arts over 50 runs. SO$_{best}$ and MMO-FSE$_{best}$ denote the best single-objective model/optimizer and the MMO-FSE with the best weight, respectively. ⊢●⊣ shows the average and standard error (SE) on the target performance objective achieved (all objective values are normalized within $[0, 1]$ and the closer to the left, the better). ⊢●⊣ denotes the best average among others. Column "$\hat{A}_{12}$ (p value)" shows the $\hat{A}_{12}$ and corrected p value when comparing the corresponding counterpart with MMO. "T", "S", "M", and "L" denotes trivial, small, medium, and large effect size, respectively. The blue cells denote MMO wins ($\hat{A}_{12} > 0.5$) while red cells mean it loses ($\hat{A}_{12} < 0.5$); otherwise it is a tie ($\hat{A}_{12} = 0.5$). Statistically significant comparisons, i.e., $\hat{A}_{12} \geq 0.56$ ($\hat{A}_{12} \leq 0.44$) and $p < 0.05$ are highlighted in bold.

	Mean/SE	$\hat{A}_{12}$ (p value)
SO$_{best}$		**.66 M (=.005)**
MMO-FSE$_{best}$		.43 M (=.114)
PMO		**.65 M (=.009)**
MMO		-

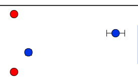
(a). MARIADB-O1

	Mean/SE	$\hat{A}_{12}$ (p value)
SO$_{best}$		.51 T (=.814)
MMO-FSE$_{best}$		.53 T (=1.33)
PMO		**.71 L (=.001)**
MMO		-

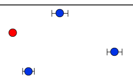
(b). MARIADB-O2

	Mean/SE	$\hat{A}_{12}$ (p value)
SO$_{best}$		.55 T (=.395)
MMO-FSE$_{best}$		.46 T (=.339)
PMO		**.66 M (<.001)**
MMO		-

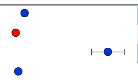
(c). STORM/WC-O1

	Mean/SE	$\hat{A}_{12}$ (p value)
SO$_{best}$		.50 T (=1.00)
MMO-FSE$_{best}$		**.99 L (<.001)**
PMO		**.66 M (=.017)**
MMO		-

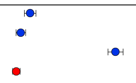
(d). STORM/WC-O2

	Mean/SE	$\hat{A}_{12}$ (p value)
SO$_{best}$		**.64 M (=.020)**
MMO-FSE$_{best}$		.44 S (=.228)
PMO		**.89 L (<.001)**
MMO		-

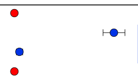
(e). VP9-O1

	Mean/SE	$\hat{A}_{12}$ (p value)
SO$_{best}$		**.64 M (=.027)**
MMO-FSE$_{best}$		.48 T (=.737)
PMO		**.87 L (<.001)**
MMO		-

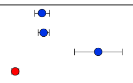
(f). VP9-O2

	Mean/SE	$\hat{A}_{12}$ (p value)
SO$_{best}$		.57 M (=.444)
MMO-FSE$_{best}$		.52 T (=.687)
PMO		**.90 L (<.001)**
MMO		-

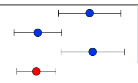
(g). STORM/RS-O1

	Mean/SE	$\hat{A}_{12}$ (p value)
SO$_{best}$		.50 T (=1.00)
MMO-FSE$_{best}$		**.94 L (<.001)**
PMO		.53 T (=1.00)
MMO		-

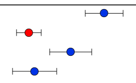
(h). STORM/RS-O2

	Mean/SE	$\hat{A}_{12}$ (p value)
SO$_{best}$		**.62 S (=.001)**
MMO-FSE$_{best}$		**.77 L (<.001)**
PMO		**.89 L (<.001)**
MMO		-

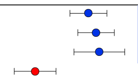
(i). LRZIP-O1

	Mean/SE	$\hat{A}_{12}$ (p value)
SO$_{best}$		.55 T (=.236)
MMO-FSE$_{best}$		**.77 L (<.001)**
PMO		**.81 L (<.001)**
MMO		-

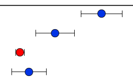
(j). LRZIP-O2

	Mean/SE	$\hat{A}_{12}$ (p value)
SO$_{best}$		**.74 L (<.001)**
MMO-FSE$_{best}$		.54 T (=.614)
PMO		.59 S (=.253)
MMO		-

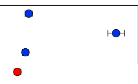
(k). MONGODB-O1

	Mean/SE	$\hat{A}_{12}$ (p value)
SO$_{best}$		**.63 S (=.031)**
MMO-FSE$_{best}$		**.64 M (=.044)**
PMO		.60 S (=.091)
MMO		-

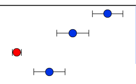
(l). MONGODB-O2

	Mean/SE	$\hat{A}_{12}$ (p value)
SO$_{best}$		**.63 S (=.023)**
MMO-FSE$_{best}$		.54 T (=.981)
PMO		.48 T (=.690)
MMO		-

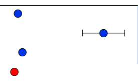
(n). KERAS-DNN/SA-O1

	Mean/SE	$\hat{A}_{12}$ (p value)
SO$_{best}$		**.62 S (=.040)**
MMO-FSE$_{best}$		**.94 L (<.001)**
PMO		**.61 S (=.049)**
MMO		-

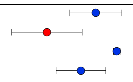
(m). KERAS-DNN/SA-O2

	Mean/SE	$\hat{A}_{12}$ (p value)
SO$_{best}$		**.65 M (=.005)**
MMO-FSE$_{best}$		.54 T (=.698)
PMO		.55 T (=.991)
MMO		-

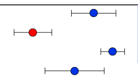
(o). KERAS-DNN/ADIAC-O1

	Mean/SE	$\hat{A}_{12}$ (p value)
SO$_{best}$		**.64 M (=.008)**
MMO-FSE$_{best}$		**.99 L (<.001)**
PMO		**.81 L (<.001)**
MMO		-

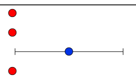
(p). KERAS-DNN/ADIAC-O2

	Mean/SE	$\hat{A}_{12}$ (p value)
SO$_{best}$		**.69 M (=.001)**
MMO-FSE$_{best}$		.45 T (=.461)
PMO		**.67 M (=.021)**
MMO		-

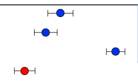
(q). X264-O1

	Mean/SE	$\hat{A}_{12}$ (p value)
SO$_{best}$		**.61 S (=.046)**
MMO-FSE$_{best}$		.52 T (=.688)
PMO		**.72 L (<.001)**
MMO		-

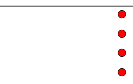
(r). X264-O2

	Mean/SE	$\hat{A}_{12}$ (p value)
SO$_{best}$		.50 T (=1.00)
MMO-FSE$_{best}$		.50 T (=1.00)
PMO		.52 T (=.135)
MMO		-

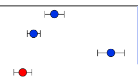
(s). LLVM-O1

	Mean/SE	$\hat{A}_{12}$ (p value)
SO$_{best}$		**.62 S (=.043)**
MMO-FSE$_{best}$		.59 S (=.182)
PMO		**.78 L (<.001)**
MMO		-

(t). LLVM-O2

	Mean/SE	$\hat{A}_{12}$ (p value)
SO$_{best}$		.50 T (=1.00)
MMO-FSE$_{best}$		.50 T (=1.00)
PMO		.50 T (=1.00)
MMO		-

(u). TRIMESH-O1

	Mean/SE	$\hat{A}_{12}$ (p value)
SO$_{best}$		**.66 M (=.017)**
MMO-FSE$_{best}$		.52 T (=.744)
PMO		**.81 L (<.001)**
MMO		-

(v). TRIMESH-O2

	% Win	% Lose	% Tie
MMO vs. SO$_{best}$	82%	0%	18%
MMO vs. MMO-FSE$_{best}$	68%	23%	9%
MMO vs. PMO	90%	5%	5%

(w). Overall % win/loss/tie for MMO versus the others based on $\hat{A}_{12}$

5 RESULTS

In this section, we present and discuss the experiment results. All code and data can be accessed at: https://github.com/ideas-labo/mmo.

5.1 RQ1: Effectiveness

5.1.1 Method

To answer **RQ1**, we compare MMO with the best state-of-the-art single-objective counterparts (as discussed in Section 4.3.1), as well as the MMO-FSE with a best-tuned weight and PMO, over all the 22 cases of study. Since the best single-objective optimizer (denoted as SO$_{best}$) and the MMO-FSE with the best weight (denoted as MMO-FSE$_{best}$) differ across the systems/objectives, we use the following procedure to select the best representative in each case:

1) Run all candidates under the full-scale experiment.
2) Rank the results using Scott-Knott test [79] according to the target performance objective[9].
3) Select the one with the best rank; if there are multiple candidates under the best rank, the one with the best average (over 50 runs) on the target performance objective would be used.

To ensure statistical significance, the statistical test[10] and effect size are reported for every pairwise comparison between our MMO and the other counterparts over 50 runs.

5.1.2 Findings

From Table 3, we can see that MMO performs considerably better than the best single-objective counterpart SO$_{best}$ (which can vary depending on the case), winning 18 out of 22 cases within which 14 of them show statistical significance[11] ($\hat{A}_{12} \geq 0.56$ and $p < 0.05$); the remaining 4 cases are all tie and there are no cases of losses. The magnitudes of gains are also clear. The improvements over PMO are also clear: MMO wins 20 cases (15 have statistically significant differences) under mostly large magnitude of gains; there are also one tie and one loss.

9. Scott-Knott test is a widely used test in SBSE [96] to distinguish different approaches into clusters based on an indicator (target performance objective in this work), between each of which are guaranteed to have statistically significant differences; the approaches within the same cluster are said to be statistically similar. The clusters are then ranked.

10. Since there are multiple comparisons, we use Kruskal-Wallis test and the corrected p values (via Holm-Bonferroni correction) of Dunn's test for all the 3 comparisons between MMO and its counterpart.

11. All Kruskal-Wallis tests show $p < 0.001$ at the global level, hence the details are omitted for simplicity of exposition.

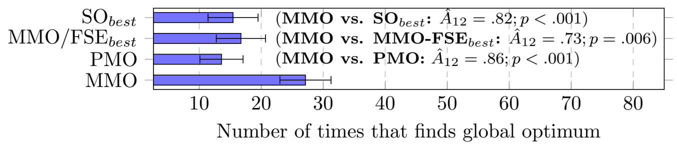


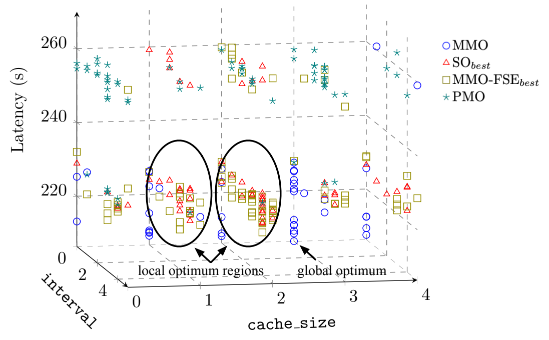
Fig. 14: The number of times to find the global optimum within 50 runs for all cases. The reported figures are the average and standard error across all cases. For all statistical comparisons with MMO, $\hat{A}_{12} > .50$ means MMO wins. All comparisons show large effect sizes.

When comparing to the MMO-FSE with the best weight (MMO-FSE$_{best}$), MMO wins 15 out of the 22 cases with 7 of them showing statistical significance; loses on 5 cases with no statistically significant ones, together with two ties. This means that, although MMO-FSE$_{best}$ is competitive, MMO can still obtain further improvement in general thanks to the new normalization method. This is especially true in some cases, such as STORM/RS-O2, where the target performance objective values are much more skewed than the auxiliary ones (recall from Figure 10). Even though MMO-FSE$_{best}$ was pre-tuned with some best weights, such finding is not surprising because: firstly, despite that the range of good weights can be reduced compared with when no normalization is used, the given set of candidate weights may not be exhaustive. Indeed, as we will show in Section 5.3, the weight tuning itself can be profoundly expensive, making exhaustive search unrealistic. As such, the chosen weight may still be far from the truly optimal weight setting. Secondly, as the population evolves, the objective values keep changing, particularly on the target performance objective. A fixed weight typically does not stay ideal during the entire evolution process. For example, the weight may be a good fit at the beginning of the evolution when the population has a relatively large range of the target performance objective values, but it may become unsuitable when the population converges into a tiny region with respect to the target performance objective.

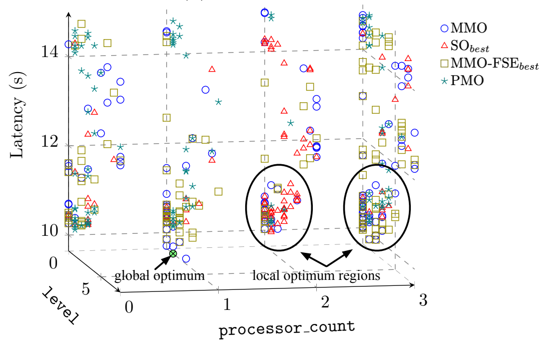
To examine whether MMO can indeed improve the chances for reaching the global optimum of the target performance objective, in Figure 14, we plot the average number of runs that each model/optimizer reaches the global optimum across all cases. We see that, as expected, MMO cannot find the global optimum for all runs under the systems studied. However, in general, it hits the global optimum more regularly than the others with statistical significance and large effect size[12].

To understand why our MMO can outperform the state-of-the-arts, we took a closer look at the configurations explored during the runs. We identified two most common patterns shown in Figure 15. As can be seen from Figure 15a, the first pattern is where MMO reaches the global optimum while the others do not; the second represents a run where the global optimum has never been found, but MMO produces a result that is much closer to it than that of the others, as shown in Figure 15b. It is worth noting that, under both patterns, there exist some large regions of local optima that cause the others to suffer more than MMO. This is

12. We use the Wilcoxon signed-rank test here since the comparisons cut across the subject systems, i.e., they are paired.



(a) MONGODB-O1



(b) LRZIP-O1

Fig. 15: Projected landscapes of the explored configurations for two exampled systems. Each point is a configuration measured in the run, regardless of whether it is preserved or not. (a) represents a case where MMO finds the global optimum while the others do not; (b) showcases the scenario where none of them found the global optimum, but MMO produces results that are much closer than those of the others.

evident by Figure 15 where the highlighted local optima regions are mostly crowded with points explored by the other counterparts. The MMO, in contrast, escapes from these local optima by exploring an even larger area while keeping the tendency towards better target performance objective, which is precisely our **Goals 1** and **2** from Section 3.

In summary, we can answer **RQ1** as:

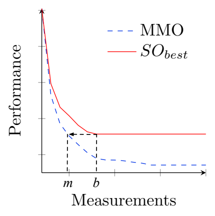
The MMO is effective because we found that
- *it provides considerably better results than the SO$_{best}$ (82%, 18 out of 22 cases) and PMO (90%, 20 out of 22 cases).*
- *it also obtains relatively good improvement over MMO-FSE$_{best}$, thanks to the new normalization.*

5.2 RQ2: Resource Efficiency

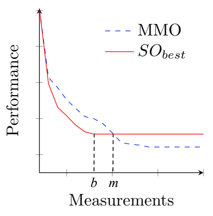
5.2.1 Method

To understand the resource efficiency of MMO in **RQ2**, for each case out of the 22, we use the following procedure:

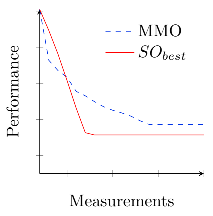
1) Identify a baseline, b, taken as the smallest number of measurements that the best single-objective counterpart (SO$_{best}$) consumes to achieve its best result of the target performance objective, averaging over 50 runs (says T).
2) For each of the others, find the smallest number of measurements, denoted as m, at which the average



(a) $s = \frac{b}{m} > 1$



(b) $s = \frac{b}{m} < 1$



(c) Failed

Fig. 16: Illustrating the calculations of speedup s.

result of the target performance objective (the mean over 50 runs) is equivalent to or better than T.

3) The speedup over SO$_{best}$, i.e., $s = \frac{b}{m}$, is reported, according to the metric used by Gao *et al.* [36].

From the example in Figure 16, we see that:

- In Figure 16a, $s > 1$ means that the approach reaches the converging performance of SO$_{best}$ by using less measurements, hence is more efficient when achieving the best of what can be produced by the baseline.
- In Figure 16b, $s < 1$ suggests that the approach reaches the converging performance of SO$_{best}$ by using more measurements, hence is less efficient even though it can lead to better results when the full budget is exhausted.
- In Figure 16c, we use $s < 0$ to denote the case where the approach has never been able to reach the converging performance of SO$_{best}$, denoted as "failed".

Of course, when $s = 1$, both approaches reach the same performance at exactly the same number of measurements.

Clearly, the greater the s, the better speedup, and hence more resources can be saved against that consumed by SO$_{best}$. In particular, if the MMO is resource-efficient, then we would expect at least $s = 1$ and ideally $s > 1$. Since in our context, the resource is the number of measurements, it reflects the time and computation required by a model. Again, we use the same SO$_{best}$ and MMO-FSE$_{best}$ from **RQ1**.

5.2.2 Findings

As can be seen from Figure 17, despite a very small number of cases where the MMO uses more resources to reach the performance level achieved by the best single-objective counterpart, most commonly it uses less number of measurements than, or at least identical to, the baseline to find the same or better results, e.g., it obtains a speedup up to 2.09×. In particular, the MMO achieves 17 cases of $s > 1$; 3 cases of $s = 1$; and 2 cases of $0 < s < 1$. Remarkably, there is no case where it fails to reach the performance level (the divided bars, denoted as $s < 0$). This indicates that the MMO overcomes local optima better and more efficiently—a key attraction to software configuration tuning due to its expensive measurements. The MMO-FSE$_{best}$ does show competitive results with respect to MMO: it has 13 cases of $s > 1$ and 4 cases of $s = 1$, but there are 5 cases of $s < 0$ due to the issues discussed in Section 3.5, which could be undesirable on certain domains. Again, the above is due to MMO covering the key properties of software configuration tuning (Section 3) while making it much less sensitive to the weight parameter.

In contrast, the PMO exhibits the worst resource efficiency in terms of the speedup over SO_{best}, as it has 3 cases of

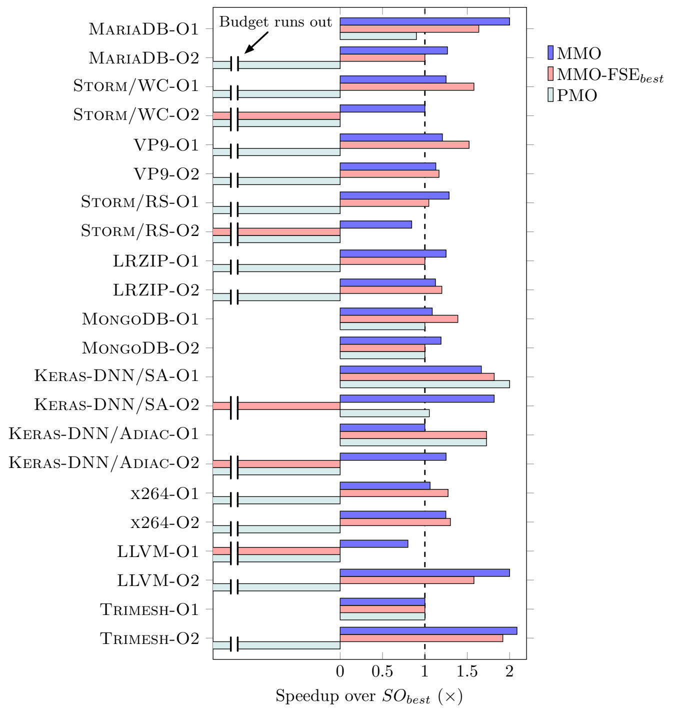


Fig. 17: Speedup (denoted as s) for MMO, MMO-FSE with the best weight (MMO$_{best}$), and the PMO for converging to the best value (the average over 50 runs) of performance objective, T, by the best single-objective counterpart (SO$_{best}$), using its budget consumption as the baseline (the dashed line at speedup 1×). $0 < s < 1$ indicating that the method is even slower (by using more measurements) than SO$_{best}$ to reach its best result, suggesting an inefficient utilization of resources. The broken bars mean T has not been reached when the tunning terminates, i.e., $s < 0$.

$s > 1$, together with 3 cases of $s = 1$; 1 case of $0 < s < 1$, respectively, while the remaining 16 cases are $s < 0$. This is a clear sign that PMO is generally resource-hungry as discussed in Section 3.

Interestingly, however, we see that PMO is considerably resource-efficient in 3 cases (KERAS-DNN/SA-O1, KERAS-DNN/SA-O1, and KERAS-DNN/ADAIC-O1). Despite being rare, this is indeed possible, because it can be severely affected by the relationship between the target and auxiliary performance objective. When there is a conflicting or weak relationship, then certainly optimizing the auxiliary performance objective would be harmful to the target one, as the valuable budget is wasted on something meaningfulless. However, when the relationship is harmonic, optimizing one can in fact beneficial to the other. In such a case, the drawback of PMO we discussed in Section 3 would become blurred. We observed that the inference time and AUC on the three cases tend to be harmonic objectives. Note that the relationship between the two objectives need not be symmetric. For example, improving inference time can also help to significantly improve AUC, but finding configurations with better AUC may slightly improve inference time. In other words, the extent of interaction can be different. This is why PMO tends to be efficient on KERAS-DNN/ADAIC-O1 but not on KERAS-DNN/ADAIC-O2.

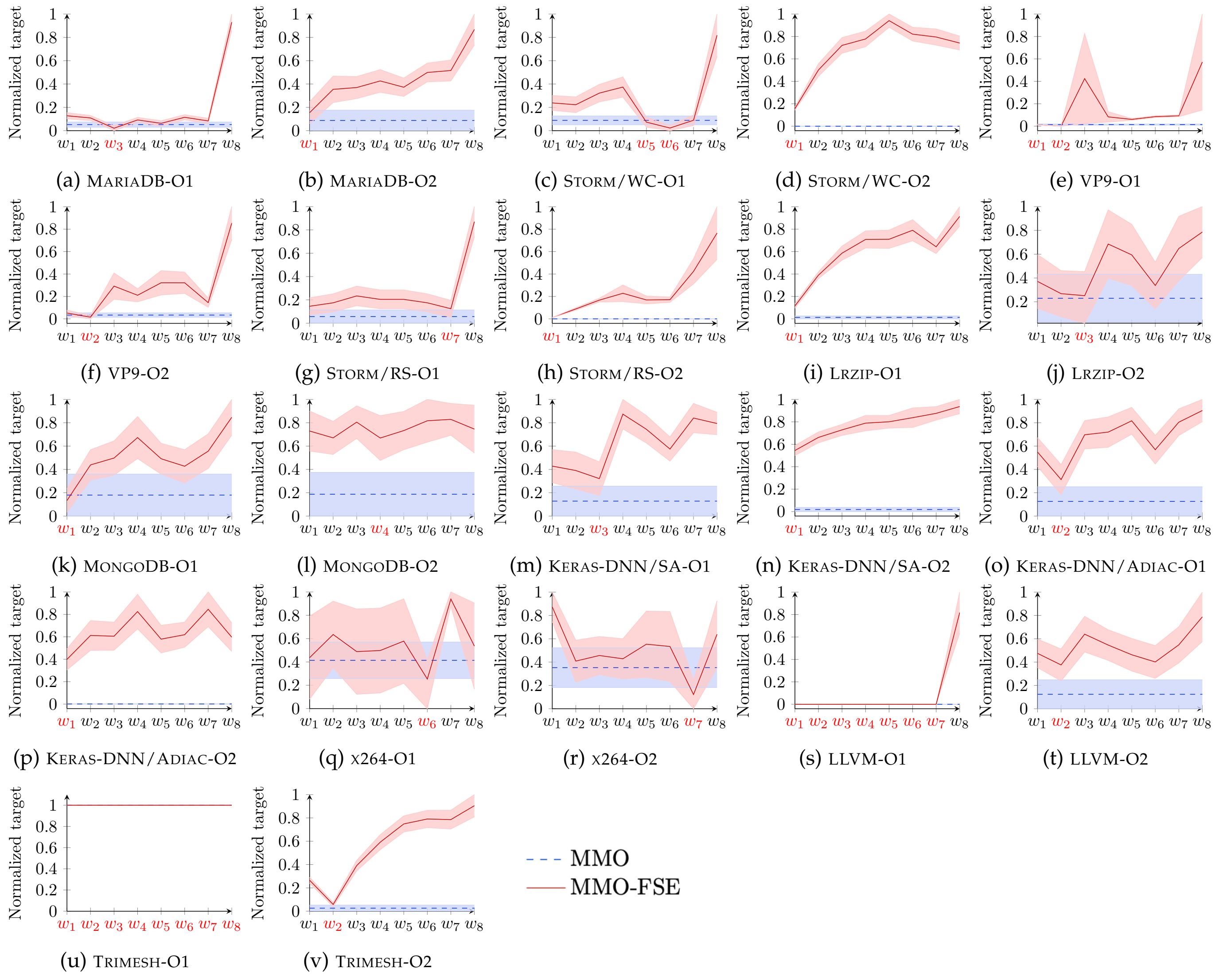


Fig. 18: Comparing MMO and MMO-FSE with different weights on the normalized target performance objective (mean and standard error) over 50 runs; the smaller, the better. $w_1, w_2, w_3, w_4, w_5, w_6, w_7, w_8$ denote weight setting of $\{0.01, 0.1, 0.3, 0.5, 0.7, 0.9, 1, 10\}$, respectively. For each case, the promising weight(s) of MMO-FSE, i.e., the one(s) that has better (or identical) mean than MMO or that has the best mean if no weight outperforms MMO, is highlighted.

As the conclusion for **RQ2**, we say:

The MMO is resource-efficient as we found that

- *it saves generally more resources than the best single-objective counterpart to reach the same or better results (for 17 out of 22 cases with up to 2.09× speedup).*
- *it leads to very competitive resource-saving and fewer cases of "failed" when compared with MMO-FSE$_{best}$.*
- *the PMO, in contrast, is much more resource-hungry.*

5.3 RQ3: Benefits over MMO-FSE

5.3.1 Method

In **RQ3**, we seek to verify the benefits provided by the weight-free design in MMO are indeed meaningful over MMO-FSE. Particularly, on each of the 22 systems/environments, we examine how MMO performs against the MMO-FSE under different weights using the full-scale experiment. Indeed, if the differences between MMO and MMO-FSE over different weights are small, then perhaps using the MMO-FSE can be sufficient. Our goal is to confirm if there are some promising weights that often lead to good enough results. As such, over 50 runs, we say a weight value is promising in MMO-FSE if:

- It leads to a result that is generally better than (or identical to) that of MMO;
- or when there are no weight values in MMO-FSE can outperform MMO overall, it is the one with the best mean result.

The other aspect we are interested in is how much extra resource would be required in order to identify at least one promising weight when using MMO-FSE. This makes sense as if the effort to find some good weights in MMO-FSE is trivial, then one would merely need to find such weight in a case-by-case manner. To investigate such, we use the following procedure in each of the 22 systems/environments:

1) Run MMO-FSE under all weights studied with an

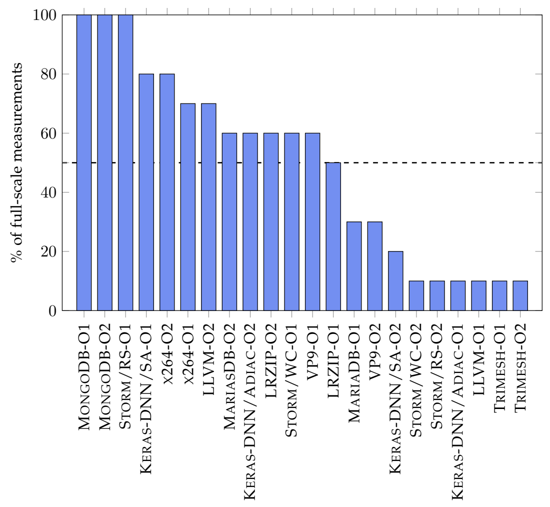


Fig. 19: The necessary resource consumed p (in terms of % on the full-scale experiments' search budget) for the MMO-FSE to find the promising weight(s) as that identified under the full-scale experiments; these are the resources that would have been saved by using MMO. As a reference, the dashed line highlights the 50% threshold of the budget. All cases are sorted in descending order.

incremental search budget that is proportional to that of the full-scale experiment, i.e., 10%, 20%, ..., 100%. The experiment under each proportion of the budget is repeated 50 runs.

2) Find the smallest proportion of the search budget, p, which discovers at least one of the promising weights as that identified previously under the full-scale experiment.
3) The p is then reported.

5.3.2 Findings

Figure 18 shows the mean on MMO and MMO-FSE under different weights; the promising ones have been highlighted. As can be seen, we observe that:

- In the majority of cases, MMO can outperform MMO-FSE over all weight settings.
- For all the cases, the performance of MMO-FSE deviates significantly under different weights.
- The promising weights often achieve considerably superior results than most of the others, i.e., up to $10\times$ better.
- It is difficult to conclude a generally promising weight over the cases. Most commonly, the promising weight can be radically diverse, e.g., it is $w = 0.01$ for MARIADB-O2 while $w = 0.9$ for X264-O1—a $90\times$ difference.

To understand how much extra resource is required to tune the weight in MMO-FSE in order to find a promising one, Figure 19 illustrates the results. Clearly, we see that for 13 out of the 22 cases, it needs 50% or more of the full-scale search budget to identify a promising weight, which may not be acceptable when using the MMO-FSE under a case; or otherwise, the quality of tuning could be compromised. Even for the 9 cases where the extra resources required are between 10% and 30%, it may still be undesirable since some

Algorithm 4: FLASH$_{\text{MMO}}$

```
Input: Configuration space $\mathcal{V}$; the system $\mathcal{F}$
Output: $s_{best}$ the best configuration on $f_t(\boldsymbol{x})$
Declare: vector of surrogates $\mathcal{M}$ (one for each performance
         objective)
1  Randomly initialize a size of $k$ configurations $\mathcal{P}$
2  MEASURE($\mathcal{P}, \mathcal{F}$)
3  $\mathcal{V} \leftarrow \mathcal{V} - \mathcal{P}$
4  while The search budget is not exhausted do
5      $\mathcal{M}$ = TRAINCARTS($\mathcal{P}$)
6      /* searching an estimated-best
          configuration in the transformed
          meta-objective space defined by the MMO
          (with NSGA-II), as shown in Algorithm 1
          */
7      ~~$o$ = FINDBESTCONFIGURATION($\mathcal{V}, \mathcal{M}$)~~
8      $o$ = MMOONNSGA-II($\mathcal{V}, \mathcal{M}$)
9      MEASURE($o, \mathcal{F}$)
10     $\mathcal{V} \leftarrow \mathcal{V} - o$
11     $\mathcal{P} \leftarrow \mathcal{P} + o$
12     if $o$ is measured to be better than $s_{best}$ on $f_t(\boldsymbol{x})$ then
13         $s_{best} = o$
14 return $s_{best}$
```

Algorithm 5: BOCA$_{\text{MMO}}$

```
Input: Configuration space $\mathcal{V}$; the system $\mathcal{F}$
Output: $s_{best}$ the best configuration on $f_t(\boldsymbol{x})$
Declare: vector of surrogates $\mathcal{M}$
1  Randomly initialize a size of $k$ configurations $\mathcal{P}$
2  MEASURE($\mathcal{P}, \mathcal{F}$)
3  while The search budget is not exhausted do
4      $\mathcal{M}$ = TRAINRANDOMFOREST($\mathcal{P}$)
5      ~~$\mathcal{I}$ = GETALLSETTINGSONIMPORTANTOPTIONS($\mathcal{M}, K$)~~
6      for ~~$i \in \mathcal{I}$~~ do
7          ~~$c$ = DECAY($j$)~~
8          ~~$\mathcal{I}$ = GETUNIMPORTANTSETTINGS($c$)~~
9          ~~$\mathcal{P}' \leftarrow$ COMBINEDSAMPLES($i, \mathcal{U}$)~~
10     /* searching an estimated-best
          configuration in the transformed
          meta-objective space defined by the MMO
          (with NSGA-II), as shown in Algorithm 1
          */
11     ~~$o$ = FINDBESTCONFIGURATION($\mathcal{P}'$)~~
12     $o$ = MMOONNSGA-II($\mathcal{V}, \mathcal{M}$)
13     MEASURE($o, \mathcal{F}$)
14     $\mathcal{P} \leftarrow \mathcal{P} + o$
15     if $o$ is measured to be better than $s_{best}$ on $f_t(\boldsymbol{x})$ then
16         $s_{best} = o$
17 return $s_{best}$
```

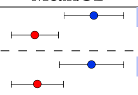
systems, such as VP9, can take up to 190 seconds to measure a single configuration.

Therefore, for **RQ3**, we say:

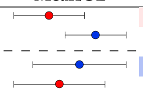
Thanks to the new normalization, the weight-free feature is meaningful in MMO as we found that

- *the MMO-FSE can be highly sensitive to the weight setting—there exist some rather diverse, case-dependent, and promising weights that perform significantly better than the others, which can only be discovered via pre-tuning.*
- *the effort required by the MMO-FSE to identify a promising weight is non-trivial, i.e., it takes 50% or more of the full-scale search budget for 13 out of 22 cases, which would otherwise be saved by using MMO instead without compromising the quality.*

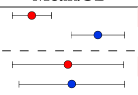
TABLE 4: Improvements on FLASH and BOCA using MMO over 50 runs. The format is the same as Table 3.

	Mean/SE	$\hat{A}_{12}$ (p value)
FLASH		**.62 S (<.001)**
FLASH$_{MMO}$		-
BOCA		**.59 S (<.001)**
BOCA$_{MMO}$		-

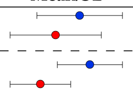
(a). MARIADB-O1

	Mean/SE	$\hat{A}_{12}$ (p value)
FLASH		.48 T (=.101)
FLASH$_{MMO}$		-
BOCA		.52 T (=.018)
BOCA$_{MMO}$		-

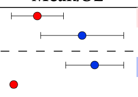
(b). MARIADB-O2

	Mean/SE	$\hat{A}_{12}$ (p value)
FLASH		**.33 M (<.001)**
FLASH$_{MMO}$		-
BOCA		.46 T (=.016)
BOCA$_{MMO}$		-

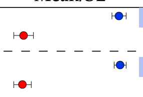
(c). STORM/WC-O1

	Mean/SE	$\hat{A}_{12}$ (p value)
FLASH		.53 T (<.001)
FLASH$_{MMO}$		-
BOCA		**.57 S (<.001)**
BOCA$_{MMO}$		-

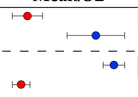
(d). STORM/WC-O2

	Mean/SE	$\hat{A}_{12}$ (p value)
FLASH		.49 T (=.530)
FLASH$_{MMO}$		-
BOCA		**.98 L (<.001)**
BOCA$_{MMO}$		-

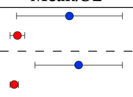
(e). VP9-O1

	Mean/SE	$\hat{A}_{12}$ (p value)
FLASH		**.91 L (<.001)**
FLASH$_{MMO}$		-
BOCA		**.95 L (<.001)**
BOCA$_{MMO}$		-

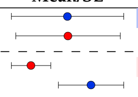
(f). VP9-O2

	Mean/SE	$\hat{A}_{12}$ (p value)
FLASH		**.31 M (<.001)**
FLASH$_{MMO}$		-
BOCA		**.88 L (<.001)**
BOCA$_{MMO}$		-

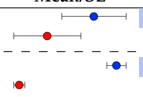
(g). STORM/RS-O1

	Mean/SE	$\hat{A}_{12}$ (p value)
FLASH		**.56 S (=.040)**
FLASH$_{MMO}$		-
BOCA		.50 T (=.015)
BOCA$_{MMO}$		-

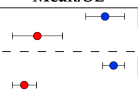
(h). STORM/RS-O2

	Mean/SE	$\hat{A}_{12}$ (p value)
FLASH		**.58 S (=.044)**
FLASH$_{MMO}$		-
BOCA		**.38 S (<.001)**
BOCA$_{MMO}$		-

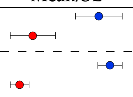
(i). LRZIP-O1

	Mean/SE	$\hat{A}_{12}$ (p value)
FLASH		**.58 S (=.015)**
FLASH$_{MMO}$		-
BOCA		**.97 L (<.001)**
BOCA$_{MMO}$		-

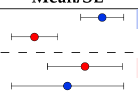
(j). LRZIP-O2

	Mean/SE	$\hat{A}_{12}$ (p value)
FLASH		**.67 M (=.008)**
FLASH$_{MMO}$		-
BOCA		**.94 L (<.001)**
BOCA$_{MMO}$		-

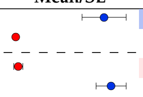
(k). MONGODB-O1

	Mean/SE	$\hat{A}_{12}$ (p value)
FLASH		**.73 L (<.001)**
FLASH$_{MMO}$		-
BOCA		**.95 L (<.001)**
BOCA$_{MMO}$		-

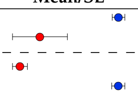
(l). MONGODB-O2

	Mean/SE	$\hat{A}_{12}$ (p value)
FLASH		**.63 S (=.011)**
FLASH$_{MMO}$		-
BOCA		.48 T (<.001)
BOCA$_{MMO}$		-

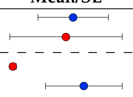
(n). KERAS-DNN/SA-O1

	Mean/SE	$\hat{A}_{12}$ (p value)
FLASH		**.78 L (<.001)**
FLASH$_{MMO}$		-
BOCA		**.03 L (<.001)**
BOCA$_{MMO}$		-

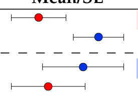
(m). KERAS-DNN/SA-O2

	Mean/SE	$\hat{A}_{12}$ (p value)
FLASH		**.61 S (=.005)**
FLASH$_{MMO}$		-
BOCA		**.05 L (<.001)**
BOCA$_{MMO}$		-

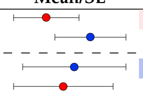
(o). KERAS-DNN/ADIAC-O1

	Mean/SE	$\hat{A}_{12}$ (p value)
FLASH		**.67 M (=.002)**
FLASH$_{MMO}$		-
BOCA		**.28 L (<.001)**
BOCA$_{MMO}$		-

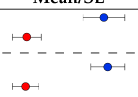
(p). KERAS-DNN/ADIAC-O2

	Mean/SE	$\hat{A}_{12}$ (p value)
FLASH		**.26 L (<.001)**
FLASH$_{MMO}$		-
BOCA		**.73 L (<.001)**
BOCA$_{MMO}$		-

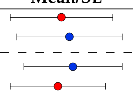
(q). X264-O1

	Mean/SE	$\hat{A}_{12}$ (p value)
FLASH		**.36 M (<.001)**
FLASH$_{MMO}$		-
BOCA		**.59 S (<.001)**
BOCA$_{MMO}$		-

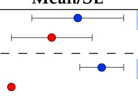
(r). X264-O2

	Mean/SE	$\hat{A}_{12}$ (p value)
FLASH		**.78 L (<.001)**
FLASH$_{MMO}$		-
BOCA		**.75 L (<.001)**
BOCA$_{MMO}$		-

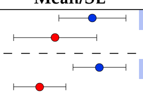
(s). LLVM-O1

	Mean/SE	$\hat{A}_{12}$ (p value)
FLASH		.47 T (<.001)
FLASH$_{MMO}$		-
BOCA		.55 T (<.001)
BOCA$_{MMO}$		-

(t). LLVM-O2

	Mean/SE	$\hat{A}_{12}$ (p value)
FLASH		.51 T (<.001)
FLASH$_{MMO}$		-
BOCA		**.63 S (<.001)**
BOCA$_{MMO}$		-

(u). TRIMESH-O1

	Mean/SE	$\hat{A}_{12}$ (p value)
FLASH		.54 T (=.103)
FLASH$_{MMO}$		-
BOCA		**.67 M (<.001)**
BOCA$_{MMO}$		-

(v). TRIMESH-O2

	% Win	% Lose	% Tie
FLASH$_{MMO}$ vs. FLASH	68%	32%	0%
BOCA$_{MMO}$ vs. BOCA	68%	27%	5%

(w). Overall % win/loss/tie for using MMO in FLASH/BOCA based on $\hat{A}_{12}$

5.4 RQ4: Consolidating Model-based Tuning

5.4.1 Method

For **RQ4**, we extended FLASH and BOCA with our MMO, denoted as FLASH$_{MMO}$ and BOCA$_{MMO}$, respectively. As shown in Algorithm 4 and 5, the change is highlighted in colors, from which we see that the amendment is merely a single line of code which changes the original search strategy that solely optimizes the target performance objective to searching over the space of MMO (working with NSGA-II). In this way, the search is conducted in the transformed meta-objective space of the surrogate-predicted objectives, in which the system $\mathcal{F}$ is replaced by the surrogates $\mathcal{M}$. For all optimizers, we allow 1,000 evaluations, including redundant ones, on the surrogate (50 population size and 20 generations in FLASH$_{MMO}$ and BOCA$_{MMO}$) as from existing work [10], [54], [76].

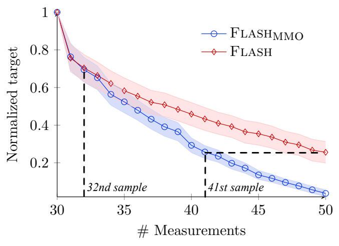
Similar to the previous sections, the statistical test[13] and effect size are reported for every pairwise comparison between the FLASH$_{MMO}$ and FLASH (BOCA$_{MMO}$ and BOCA) over 50 runs.

5.4.2 Findings

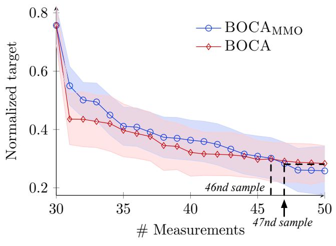
From Table 4, it is clear that FLASH$_{MMO}$ obtains better results than FLASH in general: it wins 15 out of 22 cases while loses 7 others. In particular, in those cases where FLASH$_{MMO}$ wins, 12 of them are statistically significant. In contrast, only 4 of those that it loses have $p < 0.05$ and non-trivial effect size. The relative magnitude of gains has also been significant, e.g., for VP9-O2 and STORM/RS-O2. Similar results have also been registered for comparing BOCA$_{MMO}$ and BOCA. This means that, even when searching within the surrogate-predicted space, our MMO can bring considerable improvement on model-based tuning methods like FLASH and BOCA.

To take a closer investigation, Figure 20 shows the overall search trajectories for the 20 measurements that are actually spent on tuning. Clearly, FLASH$_{MMO}$ produces a trajectory with a steeper slope than that of FLASH over all cases and runs. The standard error of the average performance is also smaller, implying that MMO can also consolidate the stability of outcomes. Of particular interesting points are at the 32*th* and 41*st* measurement: the former means that FLASH$_{MMO}$ improves the results at as little as the 2*nd* measurement into the tuning (as the first 30 are for pre-training the surrogate); while the latter reflects that the MMO helps to improve resource efficiency, achieving a $\frac{50}{41} = 1.22\times$ speedup over FLASH when reaching its best outcome under the search budget. When comparing BOCA$_{MMO}$ with BOCA, the improvement on efficiency is less obvious since BOCA leverages the information of important options. However, we see that at the 46*th* measurement, BOCA$_{MMO}$ starts to become superior to its counterpart while achieving the best

13. We use the Wilcoxon signed-rank test here since the comparisons are paired, i.e., on each run, all optimizers use the same set of randomly sampled training data for building the surrogate.



(a) on FLASH



(b) on BOCA

Fig. 20: The search trajectories on the model-based tuning methods extended by MMO and the original ones. Each point shows the average and the corresponding standard error over all cases and 50 runs. The target performance objectives are normalized and converted as minimizing objectives, if needed; hence the smaller, the better.

of BOCA at the 47*th* measurement, enabling a $\frac{50}{47} = 1.06\times$ speedup.

Although the use of a surrogate can transform the original configuration landscape into a different one according to the estimated value, the issue of local optima remains present [40], providing that the accuracy is sufficient. Therefore, the reason that improves BOCA can be attributed to the fact that MMO relieves the issue of local optima trap, as the search strategy in BOCA is restricted to a certain region of the search space with respect to the important options. As for FLASH, which is resilient to local optima due to the random search nature, the improvement is the result of MMO being able to preserve the tendency towards the best of target performance objectives, providing better guidance in the tuning.

Overall, for **RQ4**, we have:

The MMO can significantly consolidate model-based tuning methods because

- *it improves the results on both* FLASH *and* BOCA *for 15 out of 22 cases (68%), within which 12 and 13 cases are statistically significant, respectively.*
- *it enables a* $1.22\times$ *and* $1.06\times$ *speedup over* FLASH *and* BOCA*, respectively, with gradually more stable outcomes across all the cases overall.*

6 Using MMO in Practice

Using MMO in practical scenarios for any new system is straightforward. In what follows, we describe the basic steps for the application of MMO in practice:

1) Build the benchmark under which the configurable software system needs to be tuned. This is often selected from some well-known ones (e.g., WORDCOUNT for STORM) or emulated depending on the software engineers' understanding of the software's running environment.
2) Identify the key configuration options and their possible values. In practice, one can either derive these from previous studies (as we have done in this work) or based on experience and domain understanding.
3) Confirm the target performance objective and an arbitrarily chosen auxiliary performance objective, as well as how they are measured. It is important to ensure that both objectives can be influenced by different configurations. For example, when *runtime* is an important target for a system, then the auxiliary performance objective may be chosen from *CPU load*, *memory consumption*, and *energy usage*, etc.
4) Define a tuning budget in terms of the number of measurements. The parameters of the underlying NSGA-II of MMO can be chosen from some widely used ones. At this point, one can also decide on whether to use MMO as a measurement-based optimizer or a model-based one (e.g., by pairing MMO with optimizers like FLASH or BOCA). In general, the model-based version is recommended if the budget is small.
5) Run MMO on the system.

7 Threats to Validity

Threats to **internal validity** can be related to the search budget. To tackle this, we have set a budget that reaches a reasonable convergence for all the optimizers compared—a typical setup when the study aims to examine the effectiveness of mitigating local optima [74]. The other parameter settings follow what has been pragmatically used from the literature [26], [80], [31], [10], [54], [76]. or tuned through preliminary runs. However, we acknowledge that examining alternative parameters can be an interesting topic and we leave this for future work. To mitigate bias, we repeated 50 experiment runs under each case.

The metrics and evaluation used may pose threats to **construct validity**. Since there is only a single performance concern, there is no need to consider metrics with respect to multi-objective optimization [63]. We conduct the comparison based on the target performance objective and to the resource (number of measurements) required to converge to the same result. Both of these are common metrics in software configuration tuning [70], [36]. To verify statistical significance and effect size, we use the Wilcoxon test (including both the paired and non-paired versions depending on the research questions) and $\hat{A}_{12}$ to examine the results. While the MMO model is optimizer-agnostic, we examine mainly on NSGA-II in this work; using alternative multi-objective optimizers is unlikely to invalidate our conclusion but we admit the usefulness of evaluating over a wide range of optimizers with MMO, which can be part of the future work.

Threats to **external validity** can be raised from the subjects studied. We mitigated this by using 11 systems/environments that are of different domains, scales, and performance attributes, as used by prior work [70], [51], [67], [52], [75]. We also compared the proposed MMO (under the new normalization) with four state-of-the-art single-objective counterparts for software configuration tuning, PMO model, and the MMO-FSE. Further, we examine how it can help to consolidate FLASH, a recent model-based tuning method from the software engineering community. Nonetheless, we agree that studying additional systems and optimizers may prove fruitful.

8 Related Work

Broadly, optimizers for software configuration tuning can be either *measurement-based* or *model-based*.

8.1 Measurement-based Tuning

In measurement-based tuning methods, the optimizer is guided by directly measuring the configuration of the software systems. Despite the expensiveness, the measurements can accurately reflect the good or bad of a configuration (and the extents thereof). A wide range of optimizers have been studied, such as random search [8], [99], [71], [78], hill climbing [95], [62], [34], single-objective genetic algorithm [7], [80], [84] and simulated annealing [37], [42], to name a few.

Under such a single-objective model, a key difference for those optimizers lies in the tricks that attempt to overcome the issues of local optima. For example, some extend the random search to consider a wider neighboring radius of the configuration structure, hence it is more likely to jump out from the local optima [71]. Others rely on restarting from a different point, such as in restarted hill climbing, hence increasing the chance to find the "right" path from local optima to the global optimum [102], [95]. More recently, Krishna *et al.* [57] has relied on probabilistically accepting worse configurations to jump out of local optima—a typical feature of the simulated annealing [34], [42].

Our MMO differs from all the above as it lies in a higher level of abstraction—the optimization model—as opposed to the level of optimization method. In particular, with the new normalization, the purposely-crafted Pareto relation in MMO has been shown to be able to better overcome the local optima for software configuration tuning.

8.2 Model-based Tuning

Instead of solely using the measurements of software systems, the model-based tuning methods apply a surrogate (analytical [59], [30], [31] or machine learning based [70], [51], [39], [41]) to evaluate configurations, which guides the search in an optimizer. The intention is to speed up the exploration of configurations as the model evaluation is much cheaper. Yet, it has been shown that the model accuracy and the availability of initial data can become an issue [102].

Studies on model-based tuning for software systems differ mainly on the way of building surrogates and the choice of acquisition function. Among others, Jamshidi and Casale [51] use Bayesian optimization to tune software configuration, wherein the search is guided by the Gaussian Process based surrogate trained from the data collected. Nair *et al.* [70] follow a similar idea to propose FLASH, but the CART is used instead as the surrogate. More recently, Chen *et al.* [17] also follow BO and CART to propose BOCA, but they additionally identify the "important configuration options" from the Random Forest model. Such information would then inform the optimization of the acquisition function in determining what to sample next.

Since MMO lies in the level of optimization model, it is complementary to the model-based methods in which the MMO would take the surrogate values as inputs instead of the real measurements. This, as we have shown using FLASH as a case, can better consolidate the tuning results.

8.3 General Parameter Tuning

Optimizers proposed for the parameter tuning of general algorithms can also be relevant [12], [47], [9], [77], including IRace [64], ParamILS [49], SMAC [48], GGA++ [5], as well as their multi-objective variants, such as MO-ParamILS [11] and SPRINT-Race [101]. To examine a few examples, ParamILS [49] relies on iterative local search—a search procedure that may jump out of local optima using strategies similar to that of SA and SHC-r. Further, a key contribution is the capping strategy, which helps to reduce the need to measure an algorithm under some problem instances, hence saving computational resources. This is one of the goals that we seek to achieve too. Similar to Nair *et al.* [70], SMAC [48] uses Bayesian optimization but relies on a Random Forest model, which additionally considers the performance of an algorithm over a set of instances.

However, their work differs from ours in two aspects. Firstly, general algorithm configuration requires working on a set of problem instances, each coming with different features. The software configuration tuning, in contrast, is often concerned with tuning software systems under a given benchmark (i.e., one instance) [70], [51], [102], [26]. Therefore, most of their designs for saving resources (such as the capping in ParamILS) were proposed to reduce the number of instances measured. Of course, it is possible to generalize the problem to consider multiple benchmarks at the same time, yet this is outside the scope of this paper. Secondly, none of them works on the level of optimization model, and therefore, similar to the case of FLASH and BOCA, our MMO is still complementary to their optimizers.

8.4 Multi-objectivization in SBSE

Multi-objectivization, which is the notion behind our MMO model, has been applied in other SBSE problems [100], [33], [68], [86]. For example, to reproduce a crash based on the crash report, one can purposely design a new auxiliary objective, which measures how widely a test case covers the code, to be optimized alongside with the target crash distance [33]. A multi-objective optimizer, e.g., NSGA-II, is directly used thereafter. A similar case can be found also for the code refactoring problem [68]. However, during the tuning process, such a model, i.e., PMO in this paper, can result in poor resource efficiency as it wastes a significant amount of resources in optimizing the auxiliary objective, which is of no interest. This is a particularly unwelcome issue for software configuration tuning where the measurement is expensive. As we have shown in Section 5, PMO performs even worse than the classic single-objective model in most of the cases.

9 CONCLUSION AND FUTURE WORK

To mitigate the local optima issue in software configuration tuning, this paper takes a different perspective—multi-objectivizing the single objective optimization scenario. We do this by proposing a meta multi-objective model (MMO), at the level of optimization model (external part), as opposed to existing work that focuses on developing an effective single-objective optimizer (internal part). This work provides a sound analysis to interpret the principle behind MMO and explain what causes its limitation, namely eliminating the need for the weight parameter in the MMO model. Deriving on the theoretical understanding of the root cause, we then overcome such a limitation by proposing a simple yet effective new normalization.

We compare MMO under the new normalization with four state-of-the-art single-objective optimizers, the plain multi-objectivization model, and the MMO model with the old normalization from our prior FSE work over 22 cases that are of diverse performance attributes, systems, and environments. The results reveal that the MMO model:

- can generally be more effective in overcoming local optima with better results;
- and does so by utilizing resources more efficiently (better speedup) in most cases;
- saves a considerable amount of extra resources that would otherwise be required for identifying the best weight.

Furthermore, we use MMO as part of FLASH and BOCA, two recent model-based efforts from the software engineering community for configuration tuning, and revealing that it can:

- considerably consolidate the results;
- while enabling good speedup overall.

Future directions of this work are exciting and fruitful, as it paves a new way of thinking about the resolution for mitigating local optima in software configuration and perhaps in a wider context of SBSE—multi-objectivizing at the level of optimization model instead of working at the level of an optimizer/algorithm. Specifically, the most immediate next steps include extending MMO beyond two meta-objectives (e.g., through considering multiple auxiliary objectives if any) and exploring the possibility of designing a tailored multi-objectivization model for other SBSE problems. At the same time, through the analysis discussed in this work, we hope that there will be subsequent studies that take the unique characteristics of MMO into account, which can further advance software configuration tuning and beyond.

ACKNOWLEDGMENTS

This work was supported by a UKRI Grant (10054084) and a NSFC Grant (62372084). The authors wish to thank the anonymous reviewers for their highly constructive and insightful comments. Tao Chen would like to thank Yvonne and Savion for their assistance.

REFERENCES

[1] R. B. Abdessalem, A. Panichella, S. Nejati, L. C. Briand, and T. Stifter, "Automated repair of feature interaction failures in automated driving systems," in *ISSTA '20: 29th ACM SIGSOFT International Symposium on Software Testing and Analysis, Virtual Event, USA, July 18-22, 2020*. ACM, 2020, pp. 88–100. [Online]. Available: https://doi.org/10.1145/3395363.3397386

[2] H. Abdi, "Holm's sequential bonferroni procedure," *Encyclopedia of research design*, vol. 1, no. 8, pp. 1–8, 2010.

[3] M. A. Ahmad, C. Eckert, and A. Teredesai, "Interpretable machine learning in healthcare," in *Proceedings of the 2018 ACM International Conference on Bioinformatics, Computational Biology, and Health Informatics, BCB 2018, Washington, DC, USA, August 29 - September 01, 2018*. ACM, 2018, pp. 559–560. [Online]. Available: https://doi.org/10.1145/3233547.3233667

[4] M. Alrifai, T. Risse, P. Dolog, and W. Nejdl, "A scalable approach for qos-based web service selection," in *Service-Oriented Computing - ICSOC 2008 Workshops, ICSOC 2008 International Workshops, Sydney, Australia, December 1st, 2008, Revised Selected Papers*, ser. Lecture Notes in Computer Science, vol. 5472. Springer, 2008, pp. 190–199. [Online]. Available: https://doi.org/10.1007/978-3-642-01247-1_20

[5] C. Ansótegui, Y. Malitsky, H. Samulowitz, M. Sellmann, and K. Tierney, "Model-based genetic algorithms for algorithm configuration," in *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015*. AAAI Press, 2015, pp. 733–739. [Online]. Available: http://ijcai.org/Abstract/15/109

[6] A. Arcuri and L. C. Briand, "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, 2011, pp. 1–10.

[7] B. Behzad, H. V. T. Luu, J. Huchette, S. Byna, Prabhat, R. A. Aydt, Q. Koziol, and M. Snir, "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*. ACM, 2013, pp. 68:1–68:12. [Online]. Available: https://doi.org/10.1145/2503210.2503278

[8] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," *J. Mach. Learn. Res.*, vol. 13, pp. 281–305, 2012. [Online]. Available: http://dl.acm.org/citation.cfm?id=2188395

[9] L. C. T. Bezerra, M. López-Ibáñez, and T. Stützle, "Automatic configuration of multi-objective optimizers and multi-objective configuration," in *High-Performance Simulation-Based Optimization*, ser. Studies in Computational Intelligence. Springer, 2020, vol. 833, pp. 69–92. [Online]. Available: https://doi.org/10.1007/978-3-030-18764-4_4

[10] Z. Bingul, "Adaptive genetic algorithms applied to dynamic multiobjective problems," *Appl. Soft Comput.*, vol. 7, no. 3, pp. 791–799, 2007. [Online]. Available: https://doi.org/10.1016/j.asoc.2006.03.001

[11] A. Blot, H. H. Hoos, L. Jourdan, M. Kessaci-Marmion, and H. Trautmann, "Mo-paramils: A multi-objective automatic algorithm configuration framework," in *Learning and Intelligent Optimization - 10th International Conference, LION 10, Ischia, Italy, May 29 - June 1, 2016, Revised Selected Papers*, ser. Lecture Notes in Computer Science, vol. 10079. Springer, 2016, pp. 32–47. [Online]. Available: https://doi.org/10.1007/978-3-319-50349-3_3

[12] A. Blot, M. Marmion, L. Jourdan, and H. H. Hoos, "Automatic configuration of multi-objective local search algorithms for permutation problems," *Evol. Comput.*, vol. 27, no. 1, pp. 147–171, 2019. [Online]. Available: https://doi.org/10.1162/evco_a_00240

[13] L. Breiman, "Random forests," *Machine learning*, vol. 45, pp. 5–32, 2001.

[14] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone, *Classification and Regression Trees*. Wadsworth, 1984.

[15] A. Buzdalova, M. Buzdalov, and V. Parfenov, "Generation of tests for programming challenge tasks using helper-objectives," in *Search Based Software Engineering - 5th International Symposium, SSBSE 2013, St. Petersburg, Russia, August 24-26, 2013. Proceedings*, ser. Lecture Notes in Computer Science, vol. 8084. Springer, 2013, pp. 300–305. [Online]. Available: https://doi.org/10.1007/978-3-642-39742-4_28

[16] Z. Cai and Y. Wang, "A multiobjective optimization-based evolutionary algorithm for constrained optimization," *IEEE Transactions on evolutionary computation*, vol. 10, no. 6, pp. 658–675, 2006.

[17] J. Chen, N. Xu, P. Chen, and H. Zhang, "Efficient compiler autotuning via bayesian optimization," in *43rd IEEE/ACM International Conference on Software Engineering, ICSE 2021, Madrid, Spain, 22-30 May 2021*. IEEE, 2021, pp. 1198–1209. [Online]. Available: https://doi.org/10.1109/ICSE43902.2021.00110

[18] T. Chen, "All versus one: an empirical comparison on retrained and incremental machine learning for modeling performance of adaptable software," in *Proceedings of the 14th International Symposium on Software Engineering for Adaptive and Self-Managing Systems, SEAMS@ICSE 2019, Montreal, QC, Canada, May 25-31, 2019*. ACM, 2019, pp. 157–168. [Online]. Available: https://doi.org/10.1109/SEAMS.2019.00029

[19] ——, "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, 2022, pp. 78–89. [Online]. Available: https://doi.org/10.1109/SANER53432.2022.00022

[20] ——, "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*. ACM/IEEE, 2022, pp. 84–90. [Online]. Available: https://doi.org/10.1145/3524844.3528060

[21] T. Chen and R. Bahsoon, “Self-adaptive and online qos modeling for cloud-based software services,” *IEEE Trans. Software Eng.*, vol. 43, no. 5, pp. 453–475, 2017. [Online]. Available: https://doi.org/10.1109/TSE.2016.2608826

[22] ——, “Self-adaptive trade-off decision making for autoscaling cloud-based services,” *IEEE Trans. Serv. Comput.*, vol. 10, no. 4, pp. 618–632, 2017. [Online]. Available: https://doi.org/10.1109/TSC.2015.2499770

[23] T. Chen, R. Bahsoon, S. Wang, and X. Yao, “To adapt or not to adapt?: Technical debt and learning driven self-adaptation for managing runtime performance,” in *Proceedings of the 2018 ACM/SPEC International Conference on Performance Engineering, ICPE 2018, Berlin, Germany, April 09-13, 2018*. ACM, 2018, pp. 48–55. [Online]. Available: https://doi.org/10.1145/3184407.3184413

[24] T. Chen, R. Bahsoon, and X. Yao, “A survey and taxonomy of self-aware and self-adaptive cloud autoscaling systems,” *ACM Comput. Surv.*, vol. 51, no. 3, pp. 61:1–61:40, 2018. [Online]. Available: https://doi.org/10.1145/3190507

[25] ——, “Synergizing domain expertise with self-awareness in software systems: A patternized architecture guideline,” *Proc. IEEE*, vol. 108, no. 7, pp. 1094–1126, 2020. [Online]. Available: https://doi.org/10.1109/JPROC.2020.2985293

[26] T. Chen, K. Li, R. Bahsoon, and X. Yao, “FEMOSAA: Feature guided and knee driven multi-objective optimization for self-adaptive software,” *ACM Transactions on Software Engineering and Methodology*, vol. 27, no. 2, 2018.

[27] T. Chen and M. Li, “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*. ACM, 2021, pp. 453–465. [Online]. Available: https://doi.org/10.1145/3468264.3468555

[28] ——, “Do performance aspirations matter for guiding software configuration tuning? an empirical investigation under dual performance objectives,” *ACM Trans. Softw. Eng. Methodol.*, vol. 32, no. 3, pp. 68:1–68:41, 2023. [Online]. Available: https://doi.org/10.1145/3571853

[29] ——, “The weights can be harmful: Pareto search versus weighted search in multi-objective search-based software engineering,” *ACM Trans. Softw. Eng. Methodol.*, vol. 32, no. 1, pp. 5:1–5:40, 2023. [Online]. Available: https://doi.org/10.1145/3514233

[30] T. Chen, M. Li, and X. Yao, “On the effects of seeding strategies: a case for search-based multi-objective service composition,” in *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2018, Kyoto, Japan, July 15-19, 2018*. ACM, 2018, pp. 1419–1426. [Online]. Available: https://doi.org/10.1145/3205455.3205513

[31] ——, “Standing on the shoulders of giants: Seeding search-based multi-objective optimization with prior knowledge for software service composition,” *Inf. Softw. Technol.*, vol. 114, pp. 155–175, 2019. [Online]. Available: https://doi.org/10.1016/j.infsof.2019.05.013

[32] M. Cremene, M. A. Suciu, D. Pallez, and D. Dumitrescu, “Comparative analysis of multi-objective evolutionary algorithms for qos-aware web service composition,” *Appl. Soft Comput.*, vol. 39, pp. 124–139, 2016. [Online]. Available: https://doi.org/10.1016/j.asoc.2015.11.012

[33] P. Derakhshanfar, X. Devroey, A. Zaidman, A. van Deursen, and A. Panichella, “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)*, 2020.

[34] X. Ding, Y. Liu, and D. Qian, “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, 2015, pp. 831–836. [Online]. Available: https://doi.org/10.1109/ICPADS.2015.112

[35] J. J. Durillo and A. J. Nebro, “jmetal: A java framework for multi-objective optimization,” *Adv. Eng. Softw.*, vol. 42, no. 10, pp. 760–771, 2011. [Online]. Available: https://doi.org/10.1016/j.advengsoft.2011.05.014

[36] Y. Gao, Y. Zhu, H. Zhang, H. Lin, and M. Yang, “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, 2021, pp. 175–187. [Online]. Available: https://doi.org/10.1109/ICSE43902.2021.00028

[37] B. J. Garvin, M. B. Cohen, and M. B. Dwyer, “An improved meta-heuristic search for constrained interaction testing,” in *2009 1st International Symposium on Search Based Software Engineering*. IEEE, 2009, pp. 13–22.

[38] S. Gerasimou, R. Calinescu, and G. Tamburrelli, “Synthesis of probabilistic models for quality-of-service software engineering,” *Autom. Softw. Eng.*, vol. 25, no. 4, pp. 785–831, 2018. [Online]. Available: https://doi.org/10.1007/s10515-018-0235-8

[39] J. Gong and T. Chen, “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, 2022, pp. 482–494. [Online]. Available: https://doi.org/10.1145/3524842.3528431

[40] ——, “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*. ACM, 2023, pp. 858–870. [Online]. Available: https://doi.org/10.1145/3611643.3616334

[41] ——, “Predicting configuration performance in multiple environments with sequential meta-learning,” *FSE’24: Proceedings of the ACM on Software Engineering (PACMSE)*, vol. 1, no. FSE, 2024.

[42] J. Guo, Q. Yi, and A. Qasem, “Evaluating the role of optimization-specific search heuristics in effective autotuning,” *Technical report*, 2010.

[43] X. Han and T. Yu, “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, 2016, pp. 23:1–23:10. [Online]. Available: https://doi.org/10.1145/2961111.2962602

[44] M. Harman, “The current state and future of search based software engineering,” in *International Conference on Software Engineering, ISCE 2007, Workshop on the Future of Software Engineering, FOSE 2007, May 23-25, 2007, Minneapolis, MN, USA*. IEEE Computer Society, 2007, pp. 342–357. [Online]. Available: https://doi.org/10.1109/FOSE.2007.29

[45] R. M. Hierons, M. Li, X. Liu, J. A. Parejo, S. Segura, and X. Yao, “Many-objective test suite generation for software product lines,” *ACM Transactions on Software Engineering and Methodology*, vol. 29, no. 1, 2020.

[46] R. M. Hierons, M. Li, X. Liu, S. Segura, and W. Zheng, “Sip: optimal product selection from feature models using many-objective evolutionary optimization,” *ACM Transactions on Software Engineering and Methodology*, vol. 25, no. 2, p. 17, 2016.

[47] C. Huang, Y. Li, and X. Yao, “A survey of automatic parameter tuning methods for metaheuristics,” *IEEE Trans. Evol. Comput.*, vol. 24, no. 2, pp. 201–216, 2020. [Online]. Available: https://doi.org/10.1109/TEVC.2019.2921598

[48] F. Hutter, H. H. Hoos, and K. Leyton-Brown, “Sequential model-based optimization for general algorithm configuration,” in *LION5: Proc. of the 5th International Conference Learning and Intelligent Optimization*, ser. Lecture Notes in Computer Science, vol. 6683. Springer, 2011, pp. 507–523.

[49] F. Hutter, H. H. Hoos, K. Leyton-Brown, and T. Stützle, “Paramils: An automatic algorithm configuration framework,” *J. Artif. Intell. Res.*, vol. 36, pp. 267–306, 2009. [Online]. Available: https://doi.org/10.1613/jair.2861

[50] H. Ishibuchi and Y. Nojima, “Optimization of scalarizing functions through evolutionary multiobjective optimization,” in *International Conference on Evolutionary Multi-Criterion Optimization*. Springer, 2007, pp. 51–65.

[51] P. Jamshidi and G. Casale, “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, 2016, pp. 39–48.

[52] P. Jamshidi, M. Velez, C. Kästner, and N. Siegmund, “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. ACM, 2018, pp. 71–82. [Online]. Available: https://doi.org/10.1145/3236024.3236074

[53] D. R. Jones, M. Schonlau, and W. J. Welch, "Efficient global optimization of expensive black-box functions," *J. Glob. Optim.*, vol. 13, no. 4, pp. 455–492, 1998. [Online]. Available: https://doi.org/10.1023/A:1008306431147

[54] J. D. Knowles and E. J. Hughes, "Multiobjective optimization on a budget of 250 evaluations," in *Evolutionary Multi-Criterion Optimization, Third International Conference, EMO 2005, Guanajuato, Mexico, March 9-11, 2005, Proceedings*, ser. Lecture Notes in Computer Science, vol. 3410. Springer, 2005, pp. 176–190. [Online]. Available: https://doi.org/10.1007/978-3-540-31880-4_13

[55] J. D. Knowles, R. A. Watson, and D. W. Corne, "Reducing local optima in single-objective problems by multi-objectivization," in *International conference on evolutionary multi-criterion optimization*. Springer, 2001, pp. 269–283.

[56] J. Krall, T. Menzies, and M. Davies, "GALE: geometric active learning for search-based software engineering," *IEEE Trans. Software Eng.*, vol. 41, no. 10, pp. 1001–1018, 2015. [Online]. Available: https://doi.org/10.1109/TSE.2015.2432024

[57] R. Krishna, C. Tang, K. J. Sullivan, and B. Ray, "Conex: Efficient exploration of big-data system configurations for better performance," *IEEE Trans. Software Eng.*, vol. 48, no. 3, pp. 893–909, 2022. [Online]. Available: https://doi.org/10.1109/TSE.2020.3007560

[58] S. Kumar, R. Bahsoon, T. Chen, K. Li, and R. Buyya, "Multi-tenant cloud service composition using evolutionary optimization," in *24th IEEE International Conference on Parallel and Distributed Systems, ICPADS 2018, Singapore, December 11-13, 2018*. IEEE, 2018, pp. 972–979. [Online]. Available: https://doi.org/10.1109/PADSW.2018.8644640

[59] S. Kumar, T. Chen, R. Bahsoon, and R. Buyya, "DATESSO: self-adapting service composition with debt-aware two levels constraint reasoning," in *SEAMS '20: IEEE/ACM 15th International Symposium on Software Engineering for Adaptive and Self-Managing Systems, Seoul, Republic of Korea, 29 June - 3 July, 2020*. ACM, 2020, pp. 96–107. [Online]. Available: https://doi.org/10.1145/3387939.3391604

[60] K. Li, Z. Xiang, T. Chen, and K. C. Tan, "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, 2020, pp. 573–584. [Online]. Available: https://doi.org/10.1145/3324884.3416617

[61] K. Li, Z. Xiang, T. Chen, S. Wang, and K. C. Tan, "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*. ACM, 2020, pp. 566–577. [Online]. Available: https://doi.org/10.1145/3377811.3380360

[62] M. Li, L. Zeng, S. Meng, J. Tan, L. Zhang, A. R. Butt, and N. C. Fuller, "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*. ACM, 2014, pp. 165–176. [Online]. Available: https://doi.org/10.1145/2600212.2600229

[63] M. Li, T. Chen, and X. Yao, "How to evaluate solutions in pareto-based search-based software engineering? a critical review and methodological guidance," *IEEE Transactions on Software Engineering*, 2020.

[64] M. López-Ibánez, J. Dubois-Lacoste, L. P. Cáceres, M. Birattari, and T. Stützle, "The irace package: Iterated racing for automatic algorithm configuration," *Operations Research Perspectives*, vol. 3, pp. 43–58, 2016.

[65] M. Lukasiewycz, M. Glaß, F. Reimann, and J. Teich, "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*. ACM, 2011, pp. 1723–1730. [Online]. Available: https://doi.org/10.1145/2001576.2001808

[66] P. E. McKight and J. Najab, "Kruskal-wallis test," *The corsini encyclopedia of psychology*, pp. 1–1, 2010.

[67] P. Mendes, M. Casimiro, P. Romano, and D. Garlan, "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, 2020, pp. 1–8. [Online]. Available: https://doi.org/10.1109/MASCOTS50786.2020.9285971

[68] M. W. Mkaouer, M. Kessentini, S. Bechikh, and M. Ó. Cinnéide, "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*, ser. Lecture Notes in Computer Science, vol. 8636. Springer, 2014, pp. 168–183. [Online]. Available: https://doi.org/10.1007/978-3-319-09940-8_12

[69] M. W. Mkaouer, M. Kessentini, M. Ó. Cinnéide, S. Hayashi, and K. Deb, "A robust multi-objective approach to balance severity and importance of refactoring opportunities," *Empir. Softw. Eng.*, vol. 22, no. 2, pp. 894–927, 2017. [Online]. Available: https://doi.org/10.1007/s10664-016-9426-8

[70] V. Nair, Z. Yu, T. Menzies, N. Siegmund, and S. Apel, "Finding faster configurations using flash," *IEEE Transactions on Software Engineering*, vol. 46, no. 7, 2020.

[71] J. Oh, D. S. Batory, M. Myers, and N. Siegmund, "Finding near-optimal configurations in product lines by random sampling," in *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering, ESEC/FSE 2017, Paderborn, Germany, September 4-8, 2017*. ACM, 2017, pp. 61–71. [Online]. Available: https://doi.org/10.1145/3106237.3106273

[72] OpenAI, "Gpt-4 technical report," *arXiv preprint arXiv:2303.08774*, 2023.

[73] A. Panichella, F. M. Kifetew, and P. Tonella, "Reformulating branch coverage as a many-objective optimization problem," in *2015 IEEE 8th international conference on software testing, verification and validation (ICST)*. IEEE, 2015, pp. 1–10.

[74] ——, "Reformulating branch coverage as a many-objective optimization problem," in *8th IEEE International Conference on Software Testing, Verification and Validation, ICST 2015, Graz, Austria, April 13-17, 2015*. IEEE Computer Society, 2015, pp. 1–10. [Online]. Available: https://doi.org/10.1109/ICST.2015.7102604

[75] K. Peng, C. Kaltenecker, N. Siegmund, S. Apel, and T. Menzies, "VEER: disagreement-free multi-objective configuration," *CoRR*, vol. abs/2106.02716, 2021. [Online]. Available: https://arxiv.org/abs/2106.02716

[76] M. Preuss, G. Rudolph, and S. Wessing, "Tuning optimization algorithms for real-world problems by means of surrogate modeling," in *Genetic and Evolutionary Computation Conference, GECCO 2010, Proceedings, Portland, Oregon, USA, July 7-11, 2010*. ACM, 2010, pp. 401–408. [Online]. Available: https://doi.org/10.1145/1830483.1830558

[77] Y. Pushak and H. H. Hoos, "Algorithm configuration landscapes: - more benign than expected?" in *Parallel Problem Solving from Nature - PPSN XV - 15th International Conference, Coimbra, Portugal, September 8-12, 2018, Proceedings, Part II*, ser. Lecture Notes in Computer Science, vol. 11102. Springer, 2018, pp. 271–283. [Online]. Available: https://doi.org/10.1007/978-3-319-99259-4_22

[78] A. J. Ramirez, D. B. Knoester, B. H. C. Cheng, and P. K. McKinley, "Applying genetic algorithms to decision making in autonomic computing systems," in *Proceedings of the 6th International Conference on Autonomic Computing, ICAC 2009, June 15-19, 2009, Barcelona, Spain*. ACM, 2009, pp. 97–106. [Online]. Available: https://doi.org/10.1145/1555228.1555258

[79] A. J. Scott and M. Knott, "A cluster analysis method for grouping means in the analysis of variance," *Biometrics*, pp. 507–512, 1974.

[80] A. Shahbazian, S. Karthik, Y. Brun, and N. Medvidovic, "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*. ACM, 2020, pp. 1039–1051. [Online]. Available: https://doi.org/10.1145/3368089.3409749

[81] B. Shahriari, K. Swersky, Z. Wang, R. P. Adams, and N. de Freitas, "Taking the human out of the loop: A review of bayesian optimization," *Proc. IEEE*, vol. 104, no. 1, pp. 148–175, 2016. [Online]. Available: https://doi.org/10.1109/JPROC.2015.2494218

[82] N. Siegmund, S. S. Kolesnikov, C. Kästner, S. Apel, D. S. Batory, M. Rosenmüller, and G. Saake, "Predicting performance via automated feature-interaction detection," in *34th International Conference on Software Engineering, ICSE 2012, June 2-9, 2012, Zurich, Switzerland*. IEEE Computer Society, 2012, pp. 167–177. [Online]. Available: https://doi.org/10.1109/ICSE.2012.6227196

[83] R. Singh, C. Bezemer, W. Shang, and A. E. Hassan, "Optimizing the performance-related configurations of object-relational

mapping frameworks using a multi-objective genetic algorithm," in *Proceedings of the 7th ACM/SPEC International Conference on Performance Engineering, ICPE 2016, Delft, The Netherlands, March 12-16, 2016*. ACM, 2016, pp. 309–320. [Online]. Available: https://doi.org/10.1145/2851553.2851576

[84] U. Sinha, M. Cashman, and M. B. Cohen, "Using a genetic algorithm to optimize configurations in a data-driven application," in *Search-Based Software Engineering - 12th International Symposium, SSBSE 2020, Bari, Italy, October 7-8, 2020, Proceedings*, ser. Lecture Notes in Computer Science, vol. 12420. Springer, 2020, pp. 137–152. [Online]. Available: https://doi.org/10.1007/978-3-030-59762-7_10

[85] D. Sobhy, L. L. Minku, R. Bahsoon, T. Chen, and R. Kazman, "Run-time evaluation of architectures: A case study of diversification in iot," *J. Syst. Softw.*, vol. 159, 2020. [Online]. Available: https://doi.org/10.1016/j.jss.2019.110428

[86] M. Soltani, P. Derakhshanfar, A. Panichella, X. Devroey, A. Zaidman, and A. van Deursen, "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*, ser. Lecture Notes in Computer Science, vol. 11036. Springer, 2018, pp. 325–340. [Online]. Available: https://doi.org/10.1007/978-3-319-99241-9_18

[87] W. Song, Y. Wang, H.-X. Li, and Z. Cai, "Locating multiple optimal solutions of nonlinear equation systems based on multiobjective optimization," *IEEE Transactions on Evolutionary Computation*, vol. 19, no. 3, pp. 414–431, 2014.

[88] V. Steinhoff, P. Kerschke, P. Aspar, H. Trautmann, and C. Grimme, "Multiobjectivization of local search: Single-objective optimization benefits from multi-objective gradient descent," *arXiv preprint arXiv:2010.01004*, 2020.

[89] A. Strunk, "Qos-aware service composition: A survey," in *8th IEEE European Conference on Web Services (ECOWS 2010), 1-3 December 2010, Ayia Napa, Cyprus*. IEEE Computer Society, 2010, pp. 67–74. [Online]. Available: https://doi.org/10.1109/ECOWS.2010.16

[90] X. Tian, R. Han, L. Wang, G. Lu, and J. Zhan, "Latency critical big data computing in finance," *The Journal of Finance and Data Science*, vol. 1, no. 1, pp. 33–41, 2015.

[91] G. Upton and I. Cook, *A dictionary of statistics 3e*. Oxford quick reference, 2014.

[92] P. Valov, J. Petkovich, J. Guo, S. Fischmeister, and K. Czarnecki, "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*. ACM, 2017, pp. 39–50. [Online]. Available: https://doi.org/10.1145/3030207.3030216

[93] A. Vargha and H. D. Delaney, "A critique and improvement of the cl common language effect size statistics of mcgraw and wong," 2000.

[94] F. Wilcoxon, "Individual comparisons by ranking methods," 1945.

[95] B. Xi, Z. Liu, M. Raghavachari, C. H. Xia, and L. Zhang, "A smart hill-climbing algorithm for application server configuration," in *Proceedings of the 13th international conference on World Wide Web, WWW 2004, New York, NY, USA, May 17-20, 2004*. ACM, 2004, pp. 287–296. [Online]. Available: https://doi.org/10.1145/988672.988711

[96] T. Xia, R. Krishna, J. Chen, G. Mathew, X. Shen, and T. Menzies, "Hyperparameter optimization for effort estimation," *CoRR*, vol. abs/1805.00336, 2018. [Online]. Available: http://arxiv.org/abs/1805.00336

[97] T. Xu, L. Jin, X. Fan, Y. Zhou, S. Pasupathy, and R. Talwadker, "Hey, you have given me too many knobs!: understanding and dealing with over-designed configuration in system software," in *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering, ESEC/FSE 2015, Bergamo, Italy, August 30 - September 4, 2015*. ACM, 2015, pp. 307–319. [Online]. Available: https://doi.org/10.1145/2786805.2786852

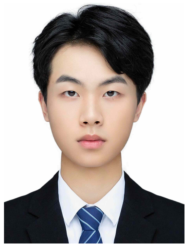
[98] Y. Xue and Y.-F. Li, "Multi-objective integer programming approaches for solving the multi-criteria test-suite minimization problem: Towards sound and complete solutions of a particular search-based software-engineering problem," *ACM Trans. Softw. Eng. Methodol.*, vol. 29, no. 3, 2020.

[99] T. Ye and S. Kalyanaraman, "A recursive random search algorithm for large-scale network parameter configuration," in *Proceedings of the International Conference on Measurements and Modeling of Computer Systems, SIGMETRICS 2003, June 9-14, 2003, San Diego, CA, USA*. ACM, 2003, pp. 196–205. [Online]. Available: https://doi.org/10.1145/781027.781052

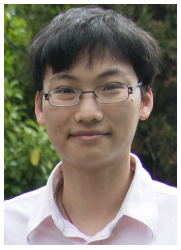
[100] Y. Yuan and W. Banzhaf, "ARJA: automated repair of java programs via multi-objective genetic programming," *IEEE Trans. Software Eng.*, vol. 46, no. 10, pp. 1040–1067, 2020. [Online]. Available: https://doi.org/10.1109/TSE.2018.2874648

[101] T. Zhang, M. Georgiopoulos, and G. C. Anagnostopoulos, "SPRINT multi-objective model racing," in *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2015, Madrid, Spain, July 11-15, 2015*. ACM, 2015, pp. 1383–1390. [Online]. Available: https://doi.org/10.1145/2739480.2754791

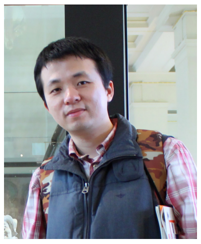
[102] Y. Zhu, J. Liu, M. Guo, Y. Bao, W. Ma, Z. Liu, K. Song, and Y. Yang, "Bestconfig: tapping the performance potential of systems via automatic configuration tuning," in *Proceedings of the 2017 Symposium on Cloud Computing, SoCC 2017, Santa Clara, CA, USA, September 24-27, 2017*. ACM, 2017, pp. 338–350. [Online]. Available: https://doi.org/10.1145/3127479.3128605



Pengzhou Chen is currently an undergraduate student in the School of Computer Science and Engineering at the University of Electronic Science and Technology of China, where he is working on the undergraduate degree and will pursue his Master degree. His research interests include performance optimization of software systems and machine learning.



Tao Chen directs the Intelligent Dependability Engineering for Adaptive Software Laboratory (IDEAS Lab), conducting cutting-edge research on the intersection between AI and Software Engineering. Currently, Tao's research interests include performance engineering, self-adaptive systems, search-based software engineering, data-driven software engineering, and their interplay with machine learning and computational intelligence. His work has been regularly published in all major Software Engineering conferences/journals (ICSE, FSE, ASE, TOSEM, and TSE) and has been supported by projects worth over £1 million from external funding bodies.



Miqing Li is an Assistant Professor in the School of Computer Science at the University of Birmingham, UK. Miqing's research sits at the interface between AI and optimization, where he is interested in developing population-based randomized algorithms (e.g., evolutionary algorithms) to solve both basic and applied (multi-objective) optimization problems. Currently, his research interests include 1) multi-objective combinatorial optimization; 2) fundamental issues in multi-objective optimization (e.g. many-objective optimization, performance assessment, multi-objectivization, fitness landscape analysis, and visualization); 3) expensive and robust optimization (EAs and Bayesian); 4) practical applications (e.g. in software engineering, machine learning, mechanical engineering, and chemical engineering).