The weights can be harmful

Chen, Tao; Li, Miqing

DOI:
10.1145/3514233

Document Version
Peer reviewed version

Citation for published version (Harvard):
Chen, T & Li, M 2023, 'The weights can be harmful: pareto search versus weighted search in multi-objective search-based software engineering', *ACM Transactions on Software Engineering and Methodology*, vol. 32, no. 1, 5, pp. 1–40. https://doi.org/10.1145/3514233

The Weights can be Harmful: Pareto Search versus Weighted Search in Multi-Objective Search-Based Software Engineering

TAO CHEN, Loughborough University, United Kingdom
MIQING LI, University of Birmingham, United Kingdom

In presence of multiple objectives to be optimized in Search-Based Software Engineering (SBSE), Pareto search has been commonly adopted. It searches for a good approximation of the problem's Pareto optimal solutions, from which the stakeholders choose the most preferred solution according to their preferences. However, when clear preferences of the stakeholders (e.g., a set of weights which reflect relative importance between objectives) are available prior to the search, weighted search is believed to be the first choice since it simplifies the search via converting the original multi-objective problem into a single-objective one and enable the search to focus on what only the stakeholders are interested in.

This paper questions such a *"weighted search first"* belief. We show that the weights can, in fact, be harmful to the search process even in the presence of clear preferences. Specifically, we conduct a large scale empirical study which consists of 38 systems/projects from three representative SBSE problems, together with two types of search budget and nine sets of weights, leading to 604 cases of comparisons. Our key finding is that weighted search reaches a certain level of solution quality by consuming relatively less resources at the early stage of the search; however, Pareto search is at the majority of the time (up to 77% of the cases) significantly better than its weighted counterpart, as long as we allow a sufficient, but not unrealistic search budget. This is a beneficial result, as it discovers a potentially new "rule-of-thumb" for the SBSE community: even when clear preferences are available, it is recommended to always consider Pareto search by default for multi-objective SBSE problems provided that solution quality is more important. Weighted search, in contrast, should only be preferred when the resource/search budget is limited, especially for expensive SBSE problems. This, together with other findings and actionable suggestions in the paper, allows us to codify pragmatic and comprehensive guidance on choosing weighted and Pareto search for SBSE under the circumstance that clear preferences are available. All code and data can be accessed at: https://github.com/ideas-labo/pareto-vs-weight-for-sbse.

CCS Concepts: • **Software and its engineering → Search-based software engineering**; *Empirical software validation*; *Software performance*.

Additional Key Words and Phrases: Search-based software engineering, multi-objective optimization, Pareto optimization, quality evaluation, quality indicator, user preference, configurable systems, adaptive systems, self-adaptive systems.

1 INTRODUCTION

Search-Based Software Engineering (SBSE) specializes the heuristic optimizers to automatically discover solutions for minimizing/maximizing objective(s) or for satisfying certain constraint(s) in various software engineering problems [41]. Over the past decades, SBSE has enjoyed a significant growth, as researches related to SBSE have spanned across different phases of software engineering, including requirements analysis [106], design [1], testing [98], deployment [20], and runtime self-adaptation [19].

Many SBSE problems involve two or more objectives, which are more or less conflicting. For example, software testing needs to make a trade-off between coverage and cost; software configuration tuning involves conflicting objectives of latency and memory consumption. It is, therefore, an important engineering decision for one to choose how the relationship between the objectives

Both authors made commensurate contributions to this research. Corresponding author: Tao Chen, t.t.chen@lboro.ac.uk. Authors' addresses: Tao Chen, t.t.chen@lboro.ac.uk, Loughborough University, United Kingdom; Miqing Li, University of Birmingham, United Kingdom, m.li.8@bham.ac.uk.

can be formulated for the search algorithm to deal with. The SBSE community takes two alternative strategies in this engineering decision-making: Pareto search or weighted search (a.k.a. utility search). The former searches for a good approximation of the Pareto front, from which the stakeholders make their choice [62, 63, 83]. The latter directly searches for a single solution that maximizes the aggregated scalar fitness of the objectives (e.g., by weighted sum [7, 41, 102]), on the basis of a set of weights (also called a weight vector) that reflects relative importance between the objectives.

In general, researchers in multi-objective SBSE choose one of the above two strategies according to availability and assumptions on the preferences of stakeholders: when no preferences (weights) between the objectives are available, Pareto search is undoubtedly chosen as it can reveal the entire Pareto front of solutions with rich diversity for one to examine without any prior information about the preferences [8, 17, 25, 106]. However, if clear preferences can be articulated, elicited, or even assumed, the weighted search would be used instead. This makes sense since naturally the weights can simplify the problem and focus the search on the direction that is only of interest to the stakeholders (no waste of the resources on searching for solutions which are not of interest to the stakeholders) [1, 2, 9, 13, 85, 91, 98]. To confirm the prevalence of weighted search under such case, we conducted a pilot search over Google Scholar with the search string "`weights`" AND "`search based software engineering`" and randomly sampled 29 papers[1] that assume weights are available between the objectives (surveys and tutorials are excluded). Among those, we found that 25 papers (i.e., 86%) have chosen weighted search over its Pareto counterpart, which is clearly a large proportion. In addition, the weighted search has also been recommended in well-known SBSE roadmaps. For example, Harman [40] pinpoints that *"where we know the relative weighting to be applied to the elements of V, we can simply use a single objective approach in which the overall fitness is a weighted sum of the predictive quality of each element in V."*

Whilst it is clear that the Pareto search strategy is a good choice when no preferences are available, the strategy is also applicable when a set of weights is given. That is, we can run a Pareto optimizer that produces a set of well-distributed solutions (approximating the Pareto front), and then apply the given weights to cherry-pick a solution therein (i.e., the solution which is the most aligned with those weights). This naturally raises a question: when there are (or assumed to have) clear preferences (i.e., weights), how does Pareto search perform in comparison with weighted search which has been believed to be well-suited to this situation in SBSE?

In SBSE, there exist some studies that have touched on the comparison between weighted search and Pareto search. For weighted search, those studies use the given weight vector to simplify the problem and guide the search, but when it comes to comparing the results returned by weighted search with those by Pareto search, they either considered generic quality indicators (e.g., hypervolume [109]) which are designed for Pareto search (such as [80, 98, 102]), or the value on every objective of the SBSE problem, e.g., [105]. Such comparisons apparently disadvantage weighted search since the stakeholders' preferences (weights) are only used in the search but not in the evaluation. That is, to evaluate weighted search under a situation that the preferences are assumed to be unavailable. This certainly results in the conclusion that Pareto search is always better than weighted search [80, 98, 102, 105]. In this work, we aim to make a more fair and comprehensive comparison between weighted search and Pareto search under clear preferences in multi-objective SBSE.

[1]Why 29? We obtained this number based on the equation of sample size by Kadam and Bhalerao [50] under the total number of papers returned by the search string (which is 1610) with 90% confidence interval.

1.1 Hypothesis and Research Questions

In this paper, we seek to understand whether Pareto search can serve as an equivalent alternative to weighted search for multi-objective SBSE problems under the circumstance that clear preferences are known. To this end, we conduct a confirmatory study, wherein our hypothesis is that:

> ***Hypothesis:*** *Pareto search may be competitive with weighted search under clear preferences, i.e., a set of weights that reflect relative importance between the objectives, if the budget is sufficient.*

The rationale for this is that the Pareto search strategy searches for the whole Pareto front while the weighted search strategy searches for a single point on that Pareto front [32, 102] — the result of the former with posterior cherry-picking can be similar to that of the latter provided that a sufficient budget is allowed. This motivates us to reconsider the validity of the *"weighted search first"* belief in multi-objective SBSE, given that it appears to be a general standard when clear preferences can be articulated, elicited or even assumed [1, 2, 7, 9, 13, 30, 41, 78, 85, 90, 98, 102].

To verify the hypothesis, we systematically compare weighted search with Pareto search in SBSE in terms of solution quality (with respect to the given weight vector) and resources required to reach its certain level. This is achieved through a comprehensive empirical study consisting of 38 instances from three representative multi-objective SBSE problems with two or three objectives, covering a wide spectrum of characteristics, representations, search space, and objectives. This, together with two types of search budget (evaluations and time) and nine (four for three objective case) weight vectors, leads to 604 cases of investigation.

The comparison is nevertheless not straightforward, since the objectives in a multi-objective SBSE often come with rather different scales. Unlike Pareto search which can be scale-free, weighted search is significantly affected by the scale of different objectives. Therefore, for using weighted search an additional decision is needed to make on how to normalize the objectives such that they become commensurable. The issue is a necessity applied to any optimizer for weighted search. While it has been shown that the best optimizer of weighted search depends on the SBSE problems and cases [41], it is not previously known whether this is also the case for commonly used normalization methods or there is indeed a best one in general. Therefore, the first research question (RQ) we wish to answer is:

> ***RQ1:*** *Is there a normalization method for weighted search that leads to the best solution in general across all multi-objective SBSE cases studied?*

To that end, we compare four normalization methods (see Section 5.1) under four optimizers (i.e., Random Search [4], Hill Climbing with restart [41], Simulated Annealing [38], and Single-Objective Genetic Algorithm [37]). These optimizers have been widely used for weighted search in multi-objective SBSE according to the well-known SBSE surveys [27, 41]. Investigating **RQ1** directly serves as the foundation to our next RQ:

> ***RQ2:*** *Given a sufficient search budget, can Pareto search produce a generally competitive solution compared with its best weighted counterpart over the multi-objective SBSE cases considered?*

Understanding **RQ2** requires us to choose a representative algorithm for Pareto and weighted search, respectively. In this work, for each case, we use the best optimizer and normalization method pair amongst the considered ones as the representative for weighted search, drawn from the results obtained in **RQ1**. Pareto search is represented by NSGA-II [28] — arguably the most commonly used Pareto optimizer in multi-objective SBSE [27, 41, 86]; and MOEA/D [104], which is an optimizer that possesses many similarities with the weighted search.

Since the preferences between the objectives are described by a weight vector, an extended question of **RQ2** for us to examine is that:

> ***RQ3:*** *Across multi-objective SBSE cases, do different weight vectors affect the comparative results between Pareto search and weighted search?*

Apart from the quality of solutions, the resources required (i.e., search budget) also plays an integral role for software engineers to decide on whether Pareto or weighted search is a preferred strategy to handle multiple objectives in SBSE. Our final RQ thus is:

> ***RQ4:*** *Overall, of Pareto search and weighted search, which is more resource efficient over different multi-objective SBSE cases?*

As mentioned, we study this on two types of search budgets that reflect the resources, i.e., the number of evaluations and time. In particular, we seek to understand which consumes less resources for reaching a certain level of solution quality.

1.2 Contributions

The findings of our empirical study are encouraging yet surprising. The most unexpected result is that *the weights can be considerably harmful to the search in multi-objective SBSE even under clear preferences*: given sufficient search budget, Pareto search is not only competitive with weighted search, but most of the time produce a significantly better solution than its weighted counterpart. Notably, a sufficient search budget does not have to be unrealistically high; rather, it is often reasonable in practice, e.g., it can be in the magnitude of seconds or less for some SBSE problems. Yet, this does not mean that weighted search can be completely abandoned: we confirm that it does consume less resources to reach a certain level of solution quality, hence it may still be preferred when the search budget is rather limited. Therefore, a key message we found from this work is that:

> ***Key message:*** *When clear preferences (weights) are available in a multi-objective SBSE problem, the choice between Pareto search and its weighted counterpart can be a trade-off between the quality of solution and the provision of search budget.*

Specifically, our contributions are:

(1) An empirical study to understand the in-depth strengths/weaknesses of both Pareto and weighted search for multi-objective SBSE under clear preference. We find that:
   - **RQ1:** The choice of the normalization methods can significantly affect the results of the weighted search and there does not exist a generally best one across all multi-objective SBSE cases. However, we do find that one often performs reasonably well (i.e., mostly the second best, if not the best) and one generally performs the worst (or the second-worst). This means that, when the objectives are of different scales, an additional process of finding the best normalization method is necessary for the weighted search to unlock its full potentials.
   - **RQ2:** Pareto search can produce a significantly better solution than its best weighted counterpart for up to 77% of the SBSE cases under sufficient search budget.
   - **RQ3:** While the gain of Pareto search over its best weighted counterpart is mostly positive across the cases, the extent of which does vary depending on the weight vector. In particular, the maximum gain often appears under a certain range of weights in an SBSE problem. On the other hand, the lowest gain often occurs when the weights are closer to extreme values, e.g., $(0.1, 0.9)$ and $(0.2, 0.8)$.

  - **RQ4:** The weighted search reaches a certain quality level by consuming less resources. However, the finding from **RQ2** suggests that it will not be able to reach the same quality level of Pareto search for most of the cases if the search is allowed to continue.
(2) Actionable suggestions derived from the findings.
(3) In-depth discussions on the reasons behind the above observations.
(4) Drawing on the findings and suggestions from the RQs, we codify pragmatic and comprehensive guidance for the SBSE practitioners to decide on whether to use Pareto search or weighted search under an SBSE situation there are clear preferences available.

To promote open science practices, all source code and data of this work can be publicly accessed at our repository: https://github.com/ideas-labo/pareto-vs-weight-for-sbse.

In what follows, this paper is organized as: Section 2 provides necessary background and ideational support of our hypothesis. Section 3 discusses the SBSE problems/instances studied and the rationale of these choices. Section 4 justifies our designs of the empirical study. Section 5 elaborates the findings, suggestions and reasons of observations, following by pragmatic guidance in Section 6. Sections 7, 8 and 9 present discussions, related work and conclusion, respectively.

2 THEORY

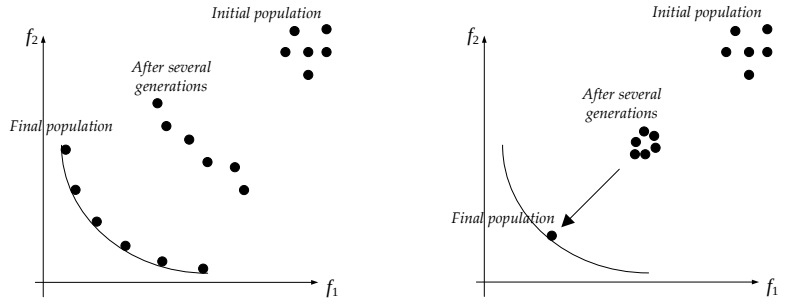
Multi-objective optimization refers to mathematical optimization involving more than one objective to be tackled simultaneously. Without loss of generality, it can be generically expressed as:

$$\min \boldsymbol{f}(x) = (f_1(x), f_2(x), \cdots, f_m(x)) \tag{1}$$

where m is the number of objectives, and x denotes a solution in the feasible solution space X, i.e., $x \in X \subset \mathbb{R}^n$ (n is the number of decision variables of the problem). As stated by Harman et al. [41], in SBSE there are two fundamental components of an optimizer that one has to specialize: (i) the representation, i.e., how x can be structured and changed; (ii) the objective function, i.e., how each single f_i can be formulated to distinguish between the good and bad solutions. In the presence of multiple objectives, a solution x^1 is said being better than x^2, called x^1 (Pareto) dominates x^2, if and only if x^1 is not worse than x^2 for all the objectives and better for at least one objective. For a solution $x \in X$, if there is no solution in X dominating x, then x is Pareto optimal. The set of all the Pareto optimal solutions is called the Pareto optimal set, which can be prohibitively large or even infinite. The image of the Pareto optimal solution set in the objective space is called the Pareto front.

2.1 Pareto Search

Since the optimum of a multi-objective problem is a Pareto front which can be infinite, a straightforward strategy to tackle the problem is to search for an approximation set that can well represent the front. Afterwards, from the approximation set, the stakeholders will choose their preferred one. This strategy is called Pareto search. In many multi-objective SBSE problems [8, 25, 106], Pareto search, working with a population-based optimizer (e.g., an evolutionary algorithm), is widely adopted, where one individual in the population is used to represent a trade-off between objectives. Note that, by Pareto search, we refer to any optimizer that treats the objectives "incomparably" and searches for the entire Pareto front. As such, it includes not only optimizers that are based on Pareto-dominance relation (e.g., NSGA-II), but also those where multiple weight vectors are used (e.g., MOEA/D [104]), weight vectors are changed during the search (e.g., AdaW [68]), or a quality indicator is used to guide the search (e.g., IBEA [108]), as long as they assume no clear preferences exist and seek to approximate the Pareto front.



(a) Pareto search (b) Weighted search

Fig. 1. Illustration of Pareto search and weighted search in a bi-objective minimization scenario, where the curve represents the Pareto front. For weighted search, the weight vector is $(0.5, 0.5)$ and the population seeks to converge into one point on the Pareto front.

2.2 Weighted Search

Another common strategy to deal with a multi-objective SBSE problem is to convert it into a single-objective problem through aggregating the objective functions (by a set of weights) [1, 2, 9, 85, 98]. For example, given a set of weights $(w_1, w_2, ..., w_m)$ which satisfies $w_1 + w_2 + ... + w_m = 1$, the multi-objective problem in Eq. (1) can be converted into minimizing the weighted sum of the objectives:

$$\min f_{ws}(x) = w_1 f_1(x) + w_2 f_2(x) + ... + w_m f_m(x) \tag{2}$$

Although how to decide such a set of weights may be arguable for SBSE problems, this strategy, as long as the weight vector can be confidently specified by the stakeholders, is commonly believed to be the best practice that can lead to the most desired result, since it incorporates the stakeholders' preferences into the search and targets one single Pareto optimal solution, thus significantly simplifying the search problem [32, 102]. Note that in a specific SBSE scenario, it is possible that the stakeholders express a clear idea about relative importance between the objectives like "`the system latency is three times more important than the memory consumption`"; then the weights for the latency and memory consumption objectives can thus be 0.75 and 0.25, respectively [7, 84, 90].

Figure 1 illustrates how the two strategies differ with respect to their search process (under a population-based optimizer, e.g., genetic algorithm).

2.3 Why Pareto Search can be Competitive to Weighted Search?

It is commonly accepted that Pareto search can be a "go-to" solution when the preferences of the stakeholders cannot be provided. Yet, we argue that even when the preferences can be confidently specified, there is still a theoretical possibility that Pareto search may outperform weighted search as well. There are two reasons supporting this. First, compared to weighted search, Pareto search may not easily get stuck in local optima, particularly when the objectives are conflicting. This is because the solutions during the search process are often incomparable (i.e., Pareto nondominated to each other) and the population-based search can maintain a set of diverse solutions. In contrast, fine-grained comparability of the scalar fitness in weighted search may not be able to maintain some solutions which can help jump out of local optima.

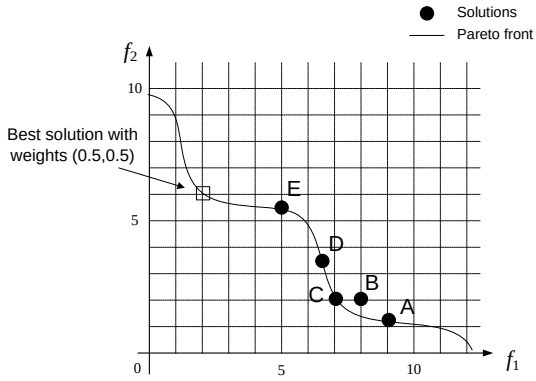


Fig. 2. An illustration of why Pareto search may be easier to find the global optimum than weighted search. Of five solutions $\mathbf{A}(9, 1.25)$, $\mathbf{B}(8, 2)$, $\mathbf{C}(7, 2)$, $\mathbf{D}(6.5, 3.5)$ and $\mathbf{E}(5, 5.5)$, $\mathbf{A}, \mathbf{C}, \mathbf{D}, \mathbf{E}$ are nondominated with each other. The solution $\mathbf{B}$ is dominated by $\mathbf{C}$ and thus is regarded as the worst solution for Pareto search. For weighted search, under the weight vector $(0.5, 0.5)$, the score of $\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D}$ and $\mathbf{E}$ is $5.125, 5, 4.5, 5$ and 5.25, respectively. Apparently, $\mathbf{E}$ has the worst score and is likely to be eliminated for weighted search. However, $\mathbf{E}$ is actually closer than the other four solutions to the possible best solution obtained for the problem under the weight vector $(0.5, 0.5)$ (i.e., the square in the figure and its score is 4). Preserving $\mathbf{E}$ is beneficial to help find the global optimum of the problem.

Consider a bi-objective minimization example shown in Figure 2 where there are five solutions $\mathbf{A}(9, 1.25)$, $\mathbf{B}(8, 2)$, $\mathbf{C}(7, 2)$, $\mathbf{D}(6.5, 3.5)$ and $\mathbf{E}(5, 5.5)$. Among them, $\mathbf{A}, \mathbf{C}, \mathbf{D}, \mathbf{E}$ are nondominated with each other, while $\mathbf{B}$ is dominated by $\mathbf{C}$. If one would like to identify the worst solution of them and eliminate it (e.g. because the population capacity is four), then $\mathbf{B}$ will be that solution for Pareto search. Now, let us say that the weight vector specified is $(0.5, 0.5)$. Then, the score of the five solutions $\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D}$ and $\mathbf{E}$ for weighted search is $5.125, 5, 4.5, 5$ and 5.25, respectively, according to $f_{ws}(x) = w_1 f_1(x) + w_2 f_2(x)$. Apparently, $\mathbf{E}$ has the worst score and is likely to be eliminated in weighted search. However, $\mathbf{E}$ is actually closer than the other four solutions to the optimal point on the Pareto front under the considered weight vector (i.e., the square in the figure and its score is 4). Apparently, removing $\mathbf{E}$ makes it harder to approach the optimal point later on during the search. In fact, the region where the solutions $\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D}$ are located can be seen as a local optimal region, and the solution $\mathbf{E}$ is outside but $\mathbf{E}$ has a worse score, thus likely to be eliminated during weighted search.

The second reason is that in weighted search the weights, which are specified to reflect the stakeholders' preferences between the objectives, may not be able to do so throughout the search. Since the objectives from most SBSE problems come with radically different scales and there are often some unknown bounds [63], the bounds of the objectives found during the search, which are used to perform the normalization of the objectives to make them comparable in the objective aggregation, can be very different from the real bounds for the problem. Working with such bounds, the weight vector, which is determined on the basis of the real bounds, may easily drive the search into some areas that are not in line with the stakeholders' preferences.

In the rest of this paper, we will check if this theoretical possibility actually occurs in real-world SBSE problems.

Table 1. Top 5 most common SBSE problems summarized from [27, 41, 63, 86].

SBSE Problem	Paper Count
Test Case Generation (TCG)	59
Next Release Problem (NRP)	28
Software Product Line Engineering (SPLE)	17
Software Configuration Tuning (SCT)	16
Web Service Composition (WSC)	16

3 SUBJECT SBSE PROBLEMS

To ensure the external validity of our empirical study, we need to choose a set of diverse SBSE problems from different domains, covering a wide spectrum of characteristics, objectives, search spaces, the dimensionality of variables, and the concrete software systems/projects. To this end, in Jan 2021, we searched over Google Scholar for well-known SBSE surveys published since 2010 with a search string "survey" AND "search based software engineering" and considered surveys that meet the following criteria:

- It covers all SBSE problems in general rather than focusing on SBSE for a particular domain. This is important to avoid bias since we are interested in finding the most common SBSE problems studied.
- It does not focus on circumstances where the weighted search is impractical. For example, many objective SBSE problem is one such case where the weights are too difficult to be specified with the increasing dimension of the objectives.
- When multiple surveys come from the same research group, only the most cited one is considered.

From the above, we identified several well-known SBSE surveys [27, 41, 63, 86], covering the SBSE papers in the past two decades. Since those surveys contain readily available classification of the SBSE papers (e.g., appendix in [41] and Table A1 in [63]), we summary the top 5 most popular problems, as shown in Table 1. Then, we read through those papers with the following selection criteria in mind:

— **Criterion 1:** To make sure our findings are meaningful, both Pareto search and weighted search should have been used by more than one paper for the SBSE problem.
— **Criterion 2:** The objectives to be optimized can be used directly to assess the quality of an optimizer for the SBSE problem, i.e., the objective is monotonically correlated with the common indicator used in the evaluation of the SBSE problem, hence the evaluation of the weighted score in weighted search is meaningful.
— **Criterion 3:** The SBSE problem should have readily available real-world software or data.

The investigation has led us to rule out the TCG problem as it violates criterion 2. This makes sense, since a higher testing coverage (a key search objective in TCG) may not necessarily detect more faults (the common metric and ultimate purpose of software testing in the evaluation). Indeed, studies [34, 92] have shown that there is a nonmonotonic and unclear correlation between the level of coverage and the number of faults detected. We also do not consider SPLE since we found no paper that aims for a weighted (or single-objective) search when addressing the problem, thereby it does not meet criterion 1. We eventually chose SCT, WSC, and NRP as the subject SBSE problems in this work, as they satisfy all the criteria above[2].

[2]Without loss of generality, we convert all maximizing objectives into minimizing one by multiplying −1.

Table 2. Software systems for SCT.

System	$\|\mathcal{O}\|$	$\|\mathcal{C}\|$	$\|\mathcal{S}\|$	Description
WC-C1-3D	Latency and throughput	3	1,343	Apache Storm with Word Count on OpenNebula (1 CPU)
WC-C3-3D	Latency and throughput	3	1,512	Apache Storm with Word Count on OpenNebula (3 CPU)
WC-C4-3D	Latency and throughput	3	756	Apache Storm with Word Count on Amazon (2 CPU)
WC-C5-5D	Latency and throughput	5	1,080	Apache Storm with Word Count on Azure (1 CPU)
RS-C3-6D	Latency and throughput	6	3,839	Apache Storm with Rolling Sort on OpenNebula (3 CPU)
WC-C1-6D	Latency and throughput	6	2,880	Apache Storm with Word Count on OpenNebula (1 CPU)
LLVM	Latency and memory	11	1,023	A compiler profiled by the standard benchmark program
TRIMESH	Latency and # Iteration	13	239,260	A library to manipulate random triangle meshes
VP8	Latency, energy and CPU load	11	2,736	A video encoder for video processing
HSQLDB	Latency, energy and CPU load	15	864	A SQL database for large volume of data

$|\mathcal{O}|$, $|\mathcal{C}|$ and $|\mathcal{S}|$ denote objectives, # configuration options and search space, respectively.

Once the SBSE problems have been identified, we read the related papers from the surveys and investigate the concrete subject systems/projects used by the most recent work. In particular, we eliminated those subjects that contain missing data or do not work as specified in the original paper. In summary, our empirical study was conducted based upon:

- 10 software systems/environments for configuration tuning used by [48, 77];
- 13 system workflows for service composition used by [24, 25, 94];
- 15 software projects/versions for planning requirements in the next release used by [35, 106].

We would like to stress that these selected problems and subjects are by no means to be comprehensive; rather, they are representative and convenient samples of SBSE problems. This is because our aim is not to exhaustively cover all SBSE problems, but as a first step to validate our hypothesis on a set of representative ones. We hope to spark a dialogue about new research opportunities regardless of whether our hypothesis can be confirmed: a positive outcome would be surprising and exciting, which encourages the SBSE community to reconsider the key criteria to choose between Pareto and weighted search in future work when weights are available (at least for the SBSE problems studied); otherwise, negative results could imply that an extended study may be required to fully confirm the invalidity of our hypothesis. In what follows, we specify the three SBSE problems in detail.

3.1 Software Configuration Tuning (SCT)

3.1.1 Problem. Many software systems are highly configurable and adaptable (at design time or runtime) [11, 12, 14, 16, 18, 20, 22, 77, 85], which raises a search problem and opportunity for one to tune their configuration options for multiple performance concerns, such as latency, throughput, and memory consumption. According to the literature, SCT has been widely studied in SBSE, e.g., [8, 12, 20–22, 48, 58, 59, 77, 85]. As mentioned, we select 10 commonly used real-world software systems and their environments from the literature [48, 77], which are specified in Table 2. As can be seen that some software systems do not involve an intractable search space; however, the solution evaluations in all of them are expensive. For example, WC-C4-3D requires several hours to explore only a small proportion of the search space [48]. This renders the exhaustive or linear search unrealistic.

3.1.2 Representation. The configuration (solution) of a software system in SCT can be represented by a vector $\overline{\mathbf{c}} = (x_1, x_2, ..., x_n)$, whereby x_n denotes the nth configuration option that can be tuned [20, 48, 77]. Since both the categorical and numeric options in configurable software can be

Table 3. System workflows for WSC.

Workflow	$\|\mathcal{O}\|$	$\|\mathcal{A}\|$	$\|\mathcal{S}\|$	Description
5AS-1	Cycle time and cost	5	1.08×10^{10}	1 parallel and 3 sequential connectors
5AS-2	Cycle time and cost	5	1.25×10^{10}	1 parallel and 2 sequential connectors
5AS-3	Cycle time and cost	5	1.73×10^{10}	4 sequential connectors
10AS-1	Cycle time and cost	10	1.23×10^{20}	3 parallel and 4 sequential connectors
10AS-2	Cycle time and cost	10	2.45×10^{20}	2 parallel and 3 sequential connectors
10AS-3	Cycle time and cost	10	2.23×10^{20}	1 parallel and 2 sequential connectors
15AS-1	Cycle time and cost	15	2.12×10^{30}	1 parallel and 2 sequential connectors
15AS-2	Cycle time and cost	15	3.17×10^{30}	4 parallel and 6 sequential connectors
15AS-3	Cycle time and cost	15	2.60×10^{30}	6 parallel and 7 sequential connectors
50AS	Cycle time and cost	50	1.86×10^{202}	10 parallel and 29 sequential connectors
5AS-3O	Cycle time, cost and latency	5	1.73×10^{10}	4 sequential connectors
10AS-3O	Cycle time, cost and latency	10	2.23×10^{20}	1 parallel and 2 sequential connectors
15AS-3O	Cycle time, cost and latency	15	2.60×10^{30}	6 parallel and 7 sequential connectors

$|\mathcal{O}|$, $|\mathcal{A}|$ and $|\mathcal{S}|$ denote the # objectives, # abstract services and search space, respectively. The objectives for all workflows are cycle time and cost. There is also a different number of abstract services in the group of each parallel connector.

discretized [20], each x_n is associated with a pre-defined list of possible values. Taking WC-C1-3D as an example, its configuration can be represented as $\overline{\mathbf{c}}$ = (`max_spout`, `spliters`, `counters`) where `max_spout`= (1, 2, ..., 1000, 10000), `spliters`= (1, 2, ..., 6) and `counters`= (1, 2, ..., 18). A particular configuration could be (1000, 5, 15).

3.1.3 Objective. As we see from Table 2, all software systems have two or three objectives to be optimized, which can be written as.

$$\text{argmax/argmin } \langle f_1(\overline{\mathbf{c}}), f_2(\overline{\mathbf{c}})\rangle \text{ or } \langle f_1(\overline{\mathbf{c}}), f_2(\overline{\mathbf{c}}), f_3(\overline{\mathbf{c}})\rangle \tag{3}$$

Albeit work exists on performance modeling for configurable and adaptable systems [13, 15**?**], there are no well-defined objective functions for f_1, f_2, and f_3 in SCT; thereby to guarantee accuracy, every evaluation needs to be done by profiling the software under a benchmark [8, 12, 21, 22, 48, 77]. For instance, optimizing LLVM involves configuring the software, running it to compile a standard benchmark program, and recording the results as the objective values thereafter. This is also the reason behind the expensiveness for SCT.

3.2 Web Service Composition (WSC)

3.2.1 Problem. Service-oriented software system is a workflow of inter-connected abstract services (e.g., via parallel or sequential connectors), each of which can be realized by a readily concrete service. The search problem is to select a set of concrete services from an explosively large pool of candidates with different performance guarantees and costs, such that the overall performance and cost of the workflow are optimized [25, 53, 94]. Such a nature of WSC is again well-fit with the purpose of SBSE, as what has been widely studied from the literature [24, 25, 54, 55, 94]. Here, we choose 13 commonly used system workflows from the existing work [24, 25] as shown in Table 3, in which the performance and cost of the candidate concrete services are sampled from the real-world dataset named WS-DREAM [107]. Their diverse numbers of abstract services and connectors result in different candidate concrete services, hence different scales of the search space.

3.2.2 Representation. In WSC, the composition (solution) of a system workflow is represented as $\overline{\mathbf{s}} = (s_1, s_2, ..., s_n)$, whereby s_n denotes the nth abstract service that needs to be realized by a concrete service [25]. For each s_n, the service broker would discover a list of m candidate concrete

services, which provide the same functionality but differ in terms of performance and cost. For example, 5AS-1 has five abstract services and thus its solution is represented as $\bar{\mathbf{s}} = (s_1, s_2, ..., s_5)$. Each abstract service may have a different number of candidate concrete services from which one needs to be selected; hence we have $s_1 = (\bar{\mathbf{s}}_{1,1}, \bar{\mathbf{s}}_{1,2}, ..., \bar{\mathbf{s}}_{1,109})$, $s_2 = (\bar{\mathbf{s}}_{2,1}, \bar{\mathbf{s}}_{2,2}, ..., \bar{\mathbf{s}}_{2,94})$, and so forth. Each concrete service is also represented as a vector of its performance and cost, e.g., if cycle time and cost of the composition are of concern, then we can have $\bar{\mathbf{s}}_{1,1} = (34s, \$14.33)$. In such case, a particular solution can be $((34, 14.33), (13, 5.42), ..., (74, 49.07))$.

3.2.3 Objective. From Table 3 we see that all system workflows seek to optimize cycle time and cost; or cycle time, cost, and latency, whose objective functions have been well-defined in the literature [24, 25, 82, 94]:

$$\text{argmin } \langle f_{time}(\bar{\mathbf{s}}), f_{cost}(\bar{\mathbf{s}}) \rangle \text{ or } \langle f_{time}(\bar{\mathbf{s}}), f_{cost}(\bar{\mathbf{s}}), f_{latency}(\bar{\mathbf{s}}) \rangle \tag{4}$$

$$f_{time}(\bar{\mathbf{s}}) = \mathbf{Max}\, T_{s_i};\ f_{cost}(\bar{\mathbf{s}}) = \sum_{i=1}^{n} C_{s_i};\ f_{latency}(\bar{\mathbf{s}}) = \sum_{i=1}^{n} L_{s_i} \tag{5}$$

whereby T_{s_i}, C_{s_i}, and L_{s_i} are the cycle time, cost, and latency of the concrete service selected for s_i, respectively; n is the number of abstract services. In essence, the overall cycle time of a workflow represents the maximum time for which a service needs to hold each request under processing. It is, therefore, often considered as the reciprocal of throughput and hence equals to the worst cycle time achieved by an abstract service (hence indicating the bottleneck). The overall cost (latency), in contrast, is the sum of cost (average delay) on all selected concrete services for the abstract services. All of them are to be minimized.

3.3 Next Release Planning (NRP)

3.3.1 Problem. As software evolves, there is often a large number of stakeholders involved and their preferences on each requirement, together with the cost of requirement fulfillment, can differ significantly [105]. Here, the search problem is what requirements should be fulfilled for the next release such that some goals, e.g., importance and cost, are optimal. The NRP problem has been widely studied in SBSE [35, 60, 105, 106], from which we choose 15 releases dataset that was mined from real-world software projects and their versions. As shown in Table 4, each software project/version involves a set of randomly sampled requirements to fulfill.

3.3.2 Representation. The representation of the release plan (solution) for NRP is a vector $\bar{\mathbf{r}} = (r_1, r_2, ..., r_n)$ where r_n is the nth requirement that can be selected to fulfill in the next software release [105, 106]. $\bar{\mathbf{r}}$ comes in a binary form and thereby r_n can only be set as either 0 or 1, meaning that r_n is not selected or selected, respectively. Considering NRP-E3, the release plan can be represented as $\bar{\mathbf{r}} = (r_1, r_2, ..., r_{47})$ and a particular one could be $(0, 1, ..., 1)$.

3.3.3 Objective. As shown in Table 4, we use two or three common objectives for all software projects/versions, namely penalty score, cost, and coverage, which have the objective functions as below [3, 35, 60, 105, 106]:

$$\text{argmin } \langle f_{score}(\bar{\mathbf{r}}), f_{cost}(\bar{\mathbf{r}}) \rangle \text{ or } \langle f_{score}(\bar{\mathbf{r}}), f_{cost}(\bar{\mathbf{r}}), f_{coverage}(\bar{\mathbf{r}}) \rangle \tag{6}$$

$$f_{score}(\bar{\mathbf{r}}) = (\sum_{i=1}^{n} \sum_{j=1}^{m} r_i \times I_j)^{-1};\ f_{cost}(\bar{\mathbf{r}}) = \sum_{i=1}^{n} r_i \times C_i;\ f_{coverage}(\bar{\mathbf{r}}) = \sigma(r_i, R_i) \tag{7}$$

where there are n requirements and m stakeholders. I_j, C_i, and R_i are respectively the satisfaction level of the ith requirement from the jth stakeholder, the related cost for fulfilling the ith requirement r_i, and the ratio of fulfilled requirement for r_i. $\sigma(\cdot)$ denotes the standard deviation across all

Table 4. Software projects/versions for NRP.

Project/Version	$\|\mathcal{O}\|$	$\|\mathcal{R}\|$	$\|\mathcal{S}\|$	Description
NRP-E1	Rank score and cost	143	1.10×10^{43}	Eclipse with 536 stakeholders
NRP-E2	Rank score and cost	123	1.06×10^{37}	Eclipse with 491 stakeholders
NRP-E3	Rank score and cost	47	1.41×10^{14}	Eclipse with 456 stakeholders
NRP-E4	Rank score and cost	139	6.97×10^{41}	Eclipse with 399 stakeholders
NRP-G1	Rank score and cost	46	2.20×10^{12}	Gnome with 445 stakeholders
NRP-G2	Rank score and cost	91	2.48×10^{27}	Gnome with 315 stakeholders
NRP-G3	Rank score and cost	102	5.07×10^{30}	Gnome with 423 stakeholders
NRP-G4	Rank score and cost	138	3.48×10^{41}	Gnome with 294 stakeholders
NRP-M1	Rank score and cost	117	1.66×10^{35}	Mozilla with 768 stakeholders
NRP-M2	Rank score and cost	78	3.02×10^{23}	Mozilla with 617 stakeholders
NRP-M3	Rank score and cost	56	7.20×10^{16}	Mozilla with 765 stakeholders
NRP-M4	Rank score and cost	140	1.39×10^{42}	Mozilla with 568 stakeholders
NRP-E-3O	Rank score, cost and coverage	47	1.41×10^{14}	Eclipse with 456 stakeholders
NRP-G-3O	Rank score, cost and coverage	46	2.20×10^{12}	Gnome with 445 stakeholders
NRP-M-3O	Rank score, cost and coverage	56	7.20×10^{16}	Mozilla with 765 stakeholders

$|\mathcal{O}|$, $|\mathcal{R}|$ and $|\mathcal{S}|$ denote the # objectives, # requirements and search space, respectively. The objectives for all projects/versions are importance score and cost.

stakeholders. As mentioned, the variable r_i can be either 0 or 1 only. All objectives are to be minimized.

4 EMPIRICAL STUDY DESIGN

We empirically investigate Pareto and weighted search on all the SBSE problems and their systems/projects from Section 3. In particular, each case of the SBSE problems is repeated 100 runs. To ensure realism, we use a cluster of machines with Intel i7 2.8GHz CPU and 8GB RAM. All experiment code was implemented in Java based on jMetal [31] and Opt4J [70]. In what follows, we will discuss the settings in greater detail.

4.1 Optimizers

Although the existing belief is to use weighted search when the weights can be explicitly given, the underlying optimizer can vary. Indeed, there is a vast set of optimizers being used for weighted search in the SBSE problems, as summarized by several surveys [23, 27, 41, 86]. To mitigate the threat to construct validity in our empirical study, we investigate four optimizers for weighted search:

- Random Search (RS);
- Hill Climbing with restart (HC);
- Simulated Annealing (SA) [38];
- Single-Objective Genetic Algorithm (SOGA) [37].

At this point, it is natural to ask why we chose those optimizers for weighted search. Indeed, since we are challenging the *"weighted search first"* belief under clear preferences, it is desired to examine all possible optimizers that have ever been used for weighted search (or single-objective search) in SBSE. However, it is fundamentally unrealistic to do so given the resource constraint [41]. Instead, we seek to examine the most widely used ones in SBSE.

To understand what are the most prevalent optimizers in SBSE, we refer to well-known SBSE surveys by Harman et al. [41] and Colanzi et al. [27]. As shown in Table 5, Colanzi et al. summarized the optimizers used in the papers published at SSBSE over the past ten years, and concluded that

Table 5. Commonality of optimizers used from [27].

Acronym	Optimizer	Paper Count
GA	(Single-Objective) Genetic Algorithm	26
NSGA-II	Non-dominated Sorting Genetic Algorithm-II	15
SA	Simulated Annealing	10
HC	Hill Climbing	7
GP	Genetic Programming	5
MOSA	Many-Objective Sorting Algorithm	4
ACO	Ant Colony Optimization	4
CP	Constraint Programming	2
IGA	Interactive Genetic Algorithm	2
LIPBS	Linearly Independent Path based Search	2
MIO	Many Independent Objective algorithm	2
SPEA2	Strength Pareto Evolutionary Algorithm	2

SOGA is the most widely used one while SA and HC are ranked as the 3rd and 4th most popular ones, respectively. Likewise, Harman et al. has also confirmed that *"the most widely used are local search, Simulated Annealing (SA), Genetic Algorithms (GAs), Genetic Programming (GP), and Hill Climbing (HC)"*. In particular, they showed that SOGA, HC, and SA are significantly more promising than the other optimizers in SBSE. The commonality of those optimizers has also been confirmed by studies of the three SBSE problems we examine [41, 48, 49]. Note that we ruled out the basic local search as HC and SA are both parts of it; we do not consider GP since it is designed to search for an optimal program rather than a solution vector that minimizes/maximizes objectives/criteria. MOSA and ACO are also ruled out as the former aims for the case with more than three objectives while the latter works better on path-finding problems. The remaining optimizers are much rarely used minorities. Although not as part of the above, we additionally take RS into account, as recommended by Arcuri and Briand [4], it should serve as a baseline for any SBSE problem.

For Pareto search, we use two optimizers:

- Non-dominated Sorting Genetic Algorithm-II (NSGA-II) [28]
- Multi-objective Evolutionary Algorithm Based on Decomposition (MOEA/D) [104]

NSGA-II is chosen because of its predominant appearance in SBSE. As shown in Table 5, Colanzi et al. [27] rank NSGA-II as the second most popular optimizers in ten year's SSBSE papers (among those for weighted/single-objective search). Similarly, Sayyad and Ammar [86] confirm that NSGA-II has been used by 53% of the papers studied — over 4× more frequent than the 2nd most popular one. The same trend has also been observed for the three SBSE problems studied in this work, as discussed in their corresponding reviews [23, 49, 106]. In contrast, despite rarely being used for SBSE, we examine MOEA/D (with a weighted sum scalar function and its dynamic bounds for normalization) because it possesses many similarities with the weighted search, as it seeks to approximate the Pareto front via internal weight vectors. Indeed, it could be fruitful if more advanced ones for Pareto search are examined. However, NSGA-II and MOEA/D, despite being developed for quite a while, have still shown their competitiveness on a lot of instances recently [61]. Moreover, if our hypothesis can be confirmed under a very basic optimizer for Pareto search, then examining more advanced ones would not change our conclusion.

Indeed, the optimizers for weighted search and those for Pareto search may have similar or rather different designs. Yet, a critical aspect that distinguishes between them is related to how the solutions are preserved into the next iterations. To give a concrete example of comparison, Algorithm 1 compares the key steps of NSGA-II and SOGA. As can be seen, although the optimizers

Algorithm 1: Unified code for NSGA-II and SOGA.

Input: Search space $\mathcal{M}$; objective functions $\overline{\mathbf{F}}$; weight vector $\overline{\mathbf{w}}$
Output: Solution set $\mathcal{S}'$ or the best solution $\mathcal{S}$

```
Randomly initialize a population of n solutions P
while The search budget is not exhausted do
    P' = ∅
    while P' < n do
        if NSGA-II then {s_x, s_y} ←MATING(P)
        else {s_x, s_y} ←MATING(w̄, P)
        {o_x, o_y} ←DOCROSSOVERANDMUTATION(M, s_x, s_y)
        EVALUATE(o_x, o_y, F̄)
        P' ← P' ⋃ {o_x, o_y}
    if NSGA-II then U ←NONDOMINATEDSORT(P ⋃ P')
    else U ←WEIGHTEDAGGREGATIONSORT(w̄, P ⋃ P')
    P ←top n solutions from U
if NSGA-II then return S' ←NONDOMINATEDSOLUTIONS(P)
else return S ←BESTWEIGHTEDSOLUTION(w̄, P)
```

can be set to use the identical mating selector, crossover, and mutation operators, there is a key difference in the criterion used in the selection processes (i.e., mating selection and surviving selection), in which the NSGA-II applies non-dominated sorting and crowding distance while SOGA sorts the solutions based on their weighted aggregation[3].

4.2 Settings

4.2.1 Components. We define the neighborhood radius in HC and SA as the solutions that differ on exactly one decision variable (e.g., a configuration option in SCT or a requirement in NRP). Such a definition of the neighborhood has been recommended by Harman [39], who states that for most SBSE problems, the neighbour in HC and SA is often a "small mutation away". Indeed, this has been widely applied in the three SBSE problems studied with promising results, e.g., [74, 97] for SCT; [52] for WSC; [6, 73] for NRP. As for SOGA, NSGA-II, and MOEA/D, the most common binary tournament is used for mating selection on all the SBSE problems [89]. For SCT and WSC, we apply the boundary mutation and uniformed crossover in all systems, as used in prior work [20, 25]. NRP differently uses bitflip mutation and one-point crossover, which are recommended in the literature [106].

4.2.2 Parameters and Search Budget. Under all SBSE problems and their systems/projects, we use the same parameter values (e.g., population size, mutation rate, and crossover rate) for both SOGA and NSGA-II, as shown in Table 6. These setting are identical to what have been commonly used from the literature [20, 25, 106]. This fits our purpose well as we intend to compare Pareto and weighted search under the most common practices.

Ideally, we would like the comparisons to be conducted on the true convergence, i.e., the best-weighted result that can be possibly achieved given an unlimited search budget. This is nevertheless not practical. Therefore, comparing them under a fixed identical search budget is more plausible. However, to avoid the outcomes of premature convergence (which is always possible) from dominating the comparisons, in this work we at least seek to allow all optimizers to reach a reasonable convergence: a degree of convergence where the best-found solution (albeit still possible to be a local optimum) does not change for some number of iterations under a search budget. In what

[3]We use the weighted sum in this work due to its prevalence [7, 41, 102].

Table 6. Parameter settings and budget.

System/Project	$\|\mathcal{M}\|$	$\|\mathcal{C}\|$	$\|\mathcal{P}\|$	$\|\mathcal{G}\|$	$\|\mathcal{E}\|$	$\|\mathcal{T}\|$
WC-C1-3D	0.1	0.9	20	30	600	49.31mins
WC-C3-3D	0.1	0.9	20	30	600	38.17mins
WC-C5-5D	0.1	0.9	20	30	600	164mins
LLVM	0.1	0.9	20	25	500	26.31mins
WC-C4-3D	0.1	0.9	20	15	300	170mins
RS-C3-6D	0.1	0.9	50	30	1.5×10^3	60.54mins
WC-C1-6D	0.1	0.9	50	30	1.5×10^3	102.98mins
TRIMESH	0.1	0.9	100	100	10^4	84.35mins
VP8	0.1	0.9	30	20	600	233.33mins
HSQLDB	0.1	0.9	50	30	1.5×10^3	81.25mins
5AS-1	0.1	0.9	100	300	3.0×10^4	2.051s
5AS-2	0.1	0.9	100	300	3.0×10^4	5.563s
5AS-3	0.1	0.9	100	300	3.0×10^4	3.59s
10AS-1	0.1	0.9	100	300	3.0×10^4	4.943s
10AS-2	0.1	0.9	100	300	3.0×10^4	3.936s
10AS-3	0.1	0.9	100	300	3.0×10^4	4.61s
15AS-1	0.1	0.9	100	300	3.0×10^4	4.027s
15AS-2	0.1	0.9	100	300	3.0×10^4	4.986s
15AS-3	0.1	0.9	100	300	3.0×10^4	4.079s
50AS	0.02	0.8	100	500	5.0×10^4	45.802s
5AS-3O	0.1	0.9	100	300	3.0×10^4	4.731s
10AS-3O	0.1	0.9	100	300	3.0×10^4	4.801s
15AS-3O	0.1	0.9	100	300	3.0×10^4	4.89s
NRP-E1	0.01	0.8	100	200	2.0×10^4	0.624s
NRP-E2	0.01	0.8	100	200	2.0×10^4	1.66s
NRP-E3	0.01	0.8	100	200	2.0×10^4	2.306s
NRP-E4	0.01	0.8	100	200	2.0×10^4	2.943s
NRP-G1	0.01	0.8	100	200	2.0×10^4	1.695s
NRP-G2	0.01	0.8	100	200	2.0×10^4	6.82s
NRP-G3	0.01	0.8	100	200	2.0×10^4	2.061s
NRP-G4	0.01	0.8	100	200	2.0×10^4	2.256s
NRP-M1	0.01	0.8	100	200	2.0×10^4	1.868s
NRP-M2	0.01	0.8	100	200	2.0×10^4	4.711s
NRP-M3	0.01	0.8	100	200	2.0×10^4	2.762s
NRP-M4	0.01	0.8	100	200	2.0×10^4	3.175s
NRP-E-3O	0.01	0.8	100	200	2.0×10^4	2.568s
NRP-G-3O	0.01	0.8	100	200	2.0×10^4	2.254s
NRP-M-3O	0.01	0.8	100	200	2.0×10^4	2.578s

$|\mathcal{M}|$, $|\mathcal{C}|$, $|\mathcal{P}|$, $|\mathcal{G}|$, $|\mathcal{E}|$, $|\mathcal{T}|$ denote the mutation rate, crossover rate, population size, # generations, # evaluations budget, and time budget, respectively.

follows, we set two metrics to represent an identical search budget under each of which Pareto and weighted search can be compared with reasonable convergence.

— **Evaluation budget:** In SBSE, using an identical number of evaluations in the comparisons is a common practice [81, 98]. To find such a fixed number of evaluations for each system/project, we firstly conduct preliminary runs on all optimizers and use the smallest evaluation number that satisfies all the following criteria:

- To ensure reasonable convergence under RS, HC, SA, and SOGA, the solution (or population) should converge to one solution point with no improvement in the last 5% of the evaluations for at least 90% of the repeated runs.

- To achieve reasonable convergence while respecting the diversity in NSGA-II, the population's solutions should change by less than 5% in the last 5% of the evaluations for at least 90% of the repeated runs. A similar setting has been used in SBSE [36].
- Each run can be completed within three hours. This is to ensure a reasonable effort and resources required for concluding the empirical study.

The results are shown in Table 6 (column $|\mathcal{E}|$).

— **Time budget:** Since the clock time is also an important factor for the practical scenarios of SBSE and an identical number of evaluations may not imply the same time consumption, in this work, we additionally compare Pareto and weighted search under an identical time budget. In particular, for each system/project, we record the longest time, t_{max}, taken by Pareto or weighted search (all optimizers) to complete one run using the fixed number of evaluations in Table 6. We then allow whichever optimizer that uses less time, if any, to run up to t_{max}. In this way, we give the ones that are originally less time-consuming a fair chance to improve (e.g., escape from premature convergence and local optima).
As can be seen from Table 6 (column $|\mathcal{T}|$), the time required to reach reasonable convergence may differ radically across the SBSE problems, due primarily to the time required to evaluate a solution — it could be up to hours-long for expensive problems like SCT, but can be as low as a few seconds (or less) for others such as WSC and NRP.

4.2.3 Possible Weight Vectors. To avoid bias to a particular assumption of stakeholders' preferences (weights), we compare Pareto and weighted search under 9 uniformly distributed sets of weights, $(0.1, 0.9)$, $(0.2, 0.8)$, ..., $(0.9, 0.1)$, for the two objective case. Such a setting covers a wide spectrum of the weights in SBSE, as what has been used from the literature [105], while being realistic enough for us to complete the empirical study. For SBSE problems with three objectives, we use three edge weight vector, $(0.1, 0.1, 0.8)$, $(0.1, 0.8, 0.1)$, and $(0.8, 0.1, 0.1)$, together with a middle one, i.e., $(0.33, 0.33, 0.33)$, as the representatives.

4.2.4 Normalization Methods. It is not uncommon to have conflicting objectives that are of radically different scales in SBSE (e.g., the latency and throughput in SCT); normalization is, therefore, crucial for weighted search to make different objectives commensurable. In this work, we consider four normalization methods that are widely used in SBSE for weighted search:

- `Dynamic`: In this method, the SBSE objectives are normalized by using their upper and lower bounds: $\frac{v-v_{min}}{v_{max}-v_{min}}$, whereby v is the raw objective value; v_{max} and v_{min} are the upper and lower bounds for that objective, respectively. However, since one may not normally know v_{max} and v_{min} from the beginning, it adopts a dynamic method wherein the objective values are normalized using the maximal and minimal values found so far as the search proceeds. This is a common method to normalize objectives when weighted search is used [7, 30, 41, 78, 90, 102].
- `Fixed`: This method is similar to `Dynamic`, but differs in the sense that the bounds are known a priori. Thus, the weights are static and no dynamic updates occur during the search. Clearly, this is only applicable to certain SBSE scenarios, as in existing work [26, 85]. When the bounds of a SBSE problem is not known naturally, in this work, we use the bounds obtained via a preliminary single-objective search that explores the extreme values of each objective.
- `Ratio`: Here, the objectives are converted by using $\frac{v}{v+1}$, whereby v is the raw objective value. This method has been used in [80, 98] .
- `None`: This is a baseline method that no normalization is applied at all, despite the fact that objectives may be of completely different scales in SBSE problems.

4.3 Analysis and Comparison

4.3.1 Metric of Solution Quality. Since there are clear preferences (weights) between the objectives, we know which solution the stakeholders favor the most, i.e., by Eq. (2). To compare the final quality of both strategies, we directly compare the weighted score, i.e., the scalar value produced by the weighted aggregation function from Eq. (2). For those that maintain a population (i.e., SOGA and NSGA-II), we use the best scalar value obtained by their population of solutions over the weight vector. To ensure an accurate comparison, we use the range of the estimated Pareto front[4] as the posterior bounds in the normalization [67]. Since we convert all objectives to be minimized, the smaller the weighted score, the better.

4.3.2 Statistical Validation. We test the statistical significance and effect size of the comparisons using:

Wilcoxon rank-sum test [100] (U-test): This was chosen because of its statistical power on pairwise comparisons [100], which fits precisely our needs. It is also a non-parametric and non-paired test that makes little assumption about the underlying distribution of the data and has been recommended in SBSE [4, 51]. In this work, we follow the common significance level as $\alpha = 0.05$.

$\hat{A}_{12}$ **effect size [93]:** We measure the pairwise effect size to evaluate the probability that one is better than the other. According to Vargha and Delaney [93], when comparing Pareto and weighted search in our experiments, $\hat{A}_{12} = 0.5$ means they are equivalent. $\hat{A}_{12} > 0.5$ and $A_{12} < 0.5$ denote that Pareto search and weighted search is better for more than 50% of the runs, respectively; they have also suggested that $\hat{A}_{12} \geq 0.56$ or $\hat{A}_{12} \leq 0.44$ are considered as non-trivial effect sizes. In particular, $0.56 \leq \hat{A}_{12} < 0.64$ (or $0.36 < \hat{A}_{12} \leq 0.44$), $0.64 \leq \hat{A}_{12} < 0.72$ (or $0.28 < \hat{A}_{12} \leq 0.36$), and $\hat{A}_{12} \geq 0.72$ (or $\hat{A}_{12} \leq 0.28$) indicate small, medium, and large effect, respectively.

Scott-Knott test [75]: Wilcoxon rank-sum test and $\hat{A}_{12}$ only work for pairwise comparison. Therefore, when comparing multiple subjects (e.g., the four normalization methods for **RQ1** and selecting the best optimizer/normalization pair of weighted search for **RQ2**-**RQ4**), we apply the Scott-Knott test — a recursive clustering based on pairwise comparisons — to rank their weighted score over 100 runs, as recommended by Mittas and Angelis [75]. In a nutshell, Scott-Knott sorts the list of treatments (the optimizers and/or normalization method) by their median weighted scores. Next, it splits the list into two sub-lists with the largest expected difference [101]. For example, suppose that we compare A, B and C, a possible split could be: $\{A, B\}$ and $\{C\}$, with the rank of 1 and 2, respectively. This means that, in the statistical sense, A and B perform similarly, but they are significantly better than C. Formally, Scott-Knott test aims to find the best split by maximizing the difference Δ in the expected mean before and after each split:

$$\Delta = \frac{|l_1|}{|l|}(\overline{l_1} - \overline{l})^2 + \frac{|l_2|}{|l|}(\overline{l_2} - \overline{l})^2 \tag{8}$$

whereby $|l_1|$ and $|l_2|$ are the sizes of two sub-lists (l_1 and l_2) from list l with a size $|l|$. $\overline{l_1}$, $\overline{l_2}$, and $\overline{l}$ denote their mean weighted score.

During the splitting, we apply a statistical hypothesis test H to check if l_1 and l_2 are significantly different. This is done by using bootstrapping and $\hat{A}_{12}$ [93] (a non-parametric effect size metric). If that is the case, Scott-Knott recurses on the splits. In other words, we divide the classifiers into different sub-lists if both bootstrap sampling and effect size test suggests that a split is statistically significant (with a confidence level of 99%) and not a small effect $\hat{A}_{12} \geq 0.56$. The sub-lists are then ranked based on their mean weighted score.

[4]The estimated Pareto front refers to the non-dominated solutions of all the solutions generated by all optimizers over all the runs (and weight vectors).

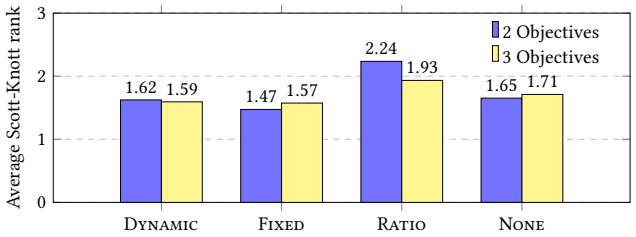


Fig. 3. The average Scott-Knott ranks over 2160 cases with 2 objectives (30 systems/projects × 4 optimizers × 9 weight vectors × 2 types of budget) and 256 cases with 3 objectives (8 systems/projects × 4 optimizers × 4 weight vectors × 2 types of budget).

In contrast to other non-parametric statistical tests that require correction on multiple comparisons (e.g., Kruskal-Wallis test), Scott-Knott test offers the following advantages:

- It does not require posterior correction, as the comparisons are essential conducted in a pairwise manner.
- It does not only show whether some treatments are statistically different or not, but also indicates which one is better than another, i.e., by means of ranking.

5 RESULTS

In this section, we present and discuss the results from our empirical study with the aim to address the questions posed in Section 1. The complete data of all cases can be found in our supplementary file: https://github.com/ideas-labo/pareto-vs-weight-for-sbse/blob/main/supplementary.pdf.

5.1 RQ1: Normalization Methods for Weighted Search

5.1.1 Method. To study **RQ1**, we compare the four normalization methods under each of the optimizers (i.e., RS, HC, SA, and SOGA) for weighted search. We do that based on all 38 systems/projects of the SBSE problems, weight vectors, and types of search budget. In each case, we use Scott-Knott test to rank the normalization methods and the weighted scores are also reported.

5.1.2 Result. As can be seen from Figure 3, it appears to be that, regardless of the number of objectives, the `Fixed` has the best Scott-Knott rank across all the cases, which is also similar to that of `Dynamic`. The reaming two, particularly the `Ratio`, performs considerably worse than the others. However, obtaining the bounds to be used in the `Fixed` method can be time-consuming when the true bounds are not naturally known beforehand.

To take a closer look at each system/project, the two objective case has been illustrated in Table 7. Here, for SCT, we see that the best normalization method varies quite differently depending on the actual software system. Yet, the worst one generally has considerably bad results and larger variation. On the results under WSC and NRP, we see quite different observations: `None` tends to be the best in general for the former while `Fixed` is often better than the other three for the latter. Their advantages are both statistically significant and to a considerable extent. `Ratio`, in contrast, often perform the worst across the systems/projects. The same trends can be observed for the three objective case, as shown in Table 8.

The above shows a clear sign that the best normalization method for weighted search highly depends on the system/project of a SBSE problem in hand, meaning that for the best result of the weighted search, an extra step is required for deciding on the best normalization method. In

Table 7. Scott-Knott ranks and the weighted scores for the normalization methods on two objectives SBSE problems. A cell summarizes 72 cases (4 algorithms × 9 weight vectors × 2 types of budget) on 100 runs each. $\mathcal{R}$ denotes the total rank over all cases, hence a 72 means that a method has been ranked the first under each case. ⊢● denotes the 25th, 50th, and 75th percentile of the weighted score. ●⊣ denotes the best median among others. The best rank for each system has been highlighted.

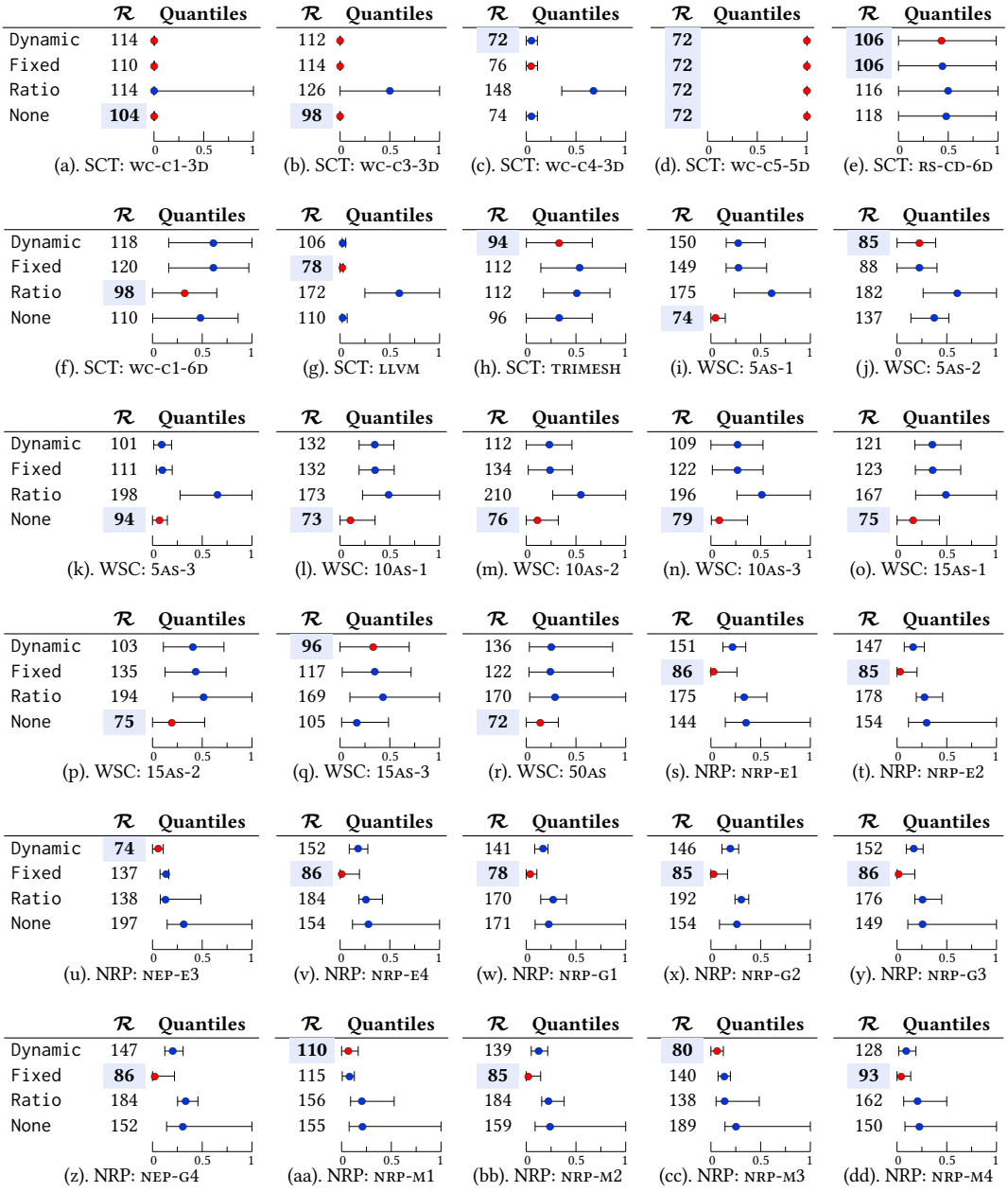
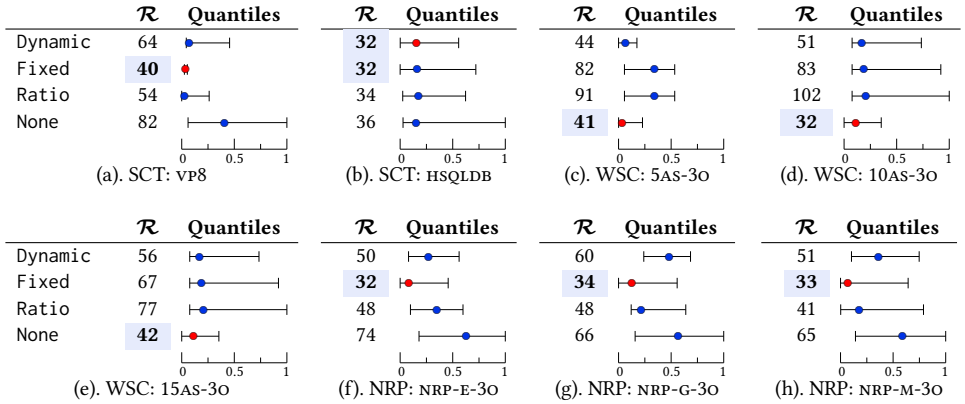


Table 8. Scott-Knott ranks and the weighted scores for the normalization methods on three objectives SBSE problems. A cell summarizes 32 cases (4 algorithms × 4 weight vectors × 2 types of budget) on 100 runs each. Formats are the same as Table 7.



contrast, this is not required for the Pareto search as it tends to be less sensitive to the objective scales.

> ***Finding:*** *There does not exist a generally best normalization for weighted search across the multi-objective SBSE problems and the systems/projects. Therefore, when necessary, an extra process of identifying the best normalization method is needed in order to achieve the best result.*

5.1.3 Implication. From the results for **RQ1**, we note that, though there is not a generally best normalization method, `Dynamic` tends to be a safe option as it often performs the second-best (if not the best) while certainly is never the worst. This matches with its overall 2nd ranking among the cases from Figure 3. However, while the `Fixed` and `None` was ranked as the 1st and 3rd on the overall ranking, the results tend to differ from that when looking at the detailed systems/projects: the former has considerably worse results than `Dynamic` for WSC, while the latter is much more inferior to `Dynamic` under NRP. In contrast, `Ratio` performs the worst in general across the systems/projects and SBSE problems, which is consistent with its overall ranking, and therefore it can be ruled out from the comparison when the resource is limited.

As such, we suggest the following for SBSE practitioners:

> ***Suggestion:*** *Do the following when using weighted search for a multi-objective SBSE problem:*
> (1) *Experimentally comparing the normalization methods (at least the four in this work) whenever the conditions permitted. The* `Ratio` *can be omitted in the case that the resource is limited.*
> (2) *When (1) is not possible, using* `Dynamic` *by default as it is a generally safer option.*

5.1.4 Reason. Clearly, the performance of the normalization method `None` highly depends on the scale of objectives in the SBSE problem. For example, in NRP the general range of the objectives in the two objective case is $[|0.0035|, |0.11|]$ and $[1.0, 504.0]$, respectively, and therefore `None` can easily lead to biased search against the first objective.

Consider the normalization methods `Ratio` and `Dynamic`. It may not be difficult to understand why `Ratio` performs worse than `Dynamic`. In contrast to `Dynamic` which always transforms the objective into a range of $[0.0, 1.0]$, `Ratio` (defined as $\frac{v}{v+1}$) is actually still affected by the scale of each individual objective — a very small or very big objective value v will squish the normalized value into a tiny part of the range $[0.0, 1.0]$. For example, the cost objective under WSC has a range of $[14.64, 122.46]$, and after transformed it will become $[0.94, 0.99]$.

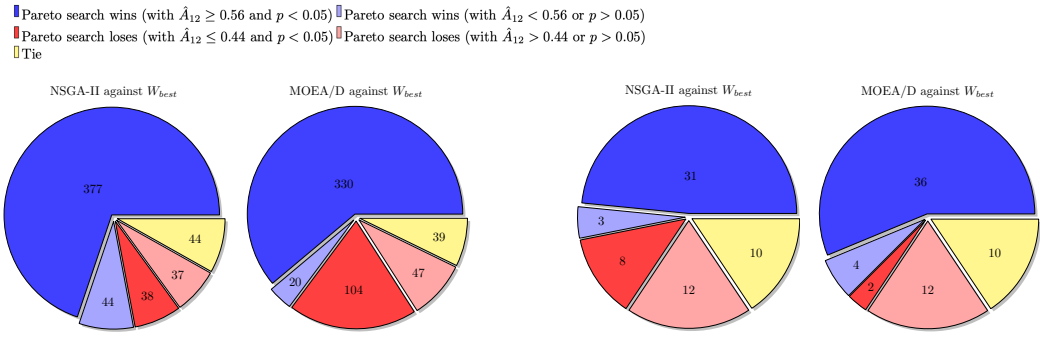
For the normalization method `Fixed`, it however may not be easy to understand why it does not always perform best, given the fact that it uses the known bounds of the SBSE problem's objectives that should have been the most accurate information. Here we use an example to explain why the `Fixed` method does not always work.

Consider a bi-objective minimization problem where the bound of the first objective is $[0.0, 1.0]$ and the bound of the second objective is $[0.0, 10.0]$, and they are known prior to the search. Let us say that the stakeholders equally like the two objectives and the solution $(0.5, 5)$ be the most preferred one. For the method `Fixed`, the weight $(10/11, 1/11)$ is chosen since the range of the second objective is ten times larger than that of the first objective. But during the search, there may just be a small portion of the space accessible, especially at the beginning stage. For example, the initial population only covers the range $[9.0, 10.0]$ on the second objective (but covers the full range $[0.0, 1.0]$ on the first objective). In this case, the weights $(10/11, 1/11)$ will likely lead to solutions close to the point $(0.0 + 1/11, 9.0 + 10/11)$ to be preferred. This will drive the search away from the region of the desired solution $(0.5, 5)$. In multi-objective SBSE, it is not uncommon that the accessibility of the objectives is different during the search. Take WSC as an example, the value range of cycle time is less accessible compared with that of the cost (and latency). The is due to the way of how the objectives are calculated determining that the cycle time is much less sensitive to different compositions than the cost (and latency), as only the concrete service with the maximum cycle time would be used while the cost is always the summation of all.

5.2 RQ2: Quality of Solution

5.2.1 Method. To answer **RQ2**, we perform pairwise comparisons between Pareto and weighted search using all the 38 systems/projects of the SBSE problems, the nine sets of weights, and the two types of search budget (evaluation and time), leading to a total of 604 cases of investigation (540 for two objective and 64 for three objective case). In each of those cases, we extract the best optimizer (i.e., amongst RS, HC, SA, and SOGA) and its normalization method (i.e., amongst `Dynamic`, `Fixed`, `Ratio`, and `None`), denoted as W_{best}, to compare with the NSGA-II and MOEA/D independently. To identify the best, we leverage on the result from **RQ1** to find the one from the best Scott-Knott rank; if there are multiple optimizers (and their normalization methods) in the best rank, we use the pair with the best median (and smallest IQR, if needed) weighted score as the best. This makes sense since we are only interested in the result of the best optimizer/normalization method for the weighted search against that of the Pareto counterpart. For each case, both Wilcoxon rank-sum test and $\hat{A}_{12}$ are used (over 100 runs) to test the significance of the comparison on the resulted weighted score between W_{best} and NSGA-II (or MOEA/D).

5.2.2 Result. As we can see from Figure 4, for two objective case, Pareto search wins on 77% of the cases (421 out of 540) for NSGA-II and 65% of the cases (350 out of 540) for MOEA/D; loses on 14% cases (75 out of 540) for NSGA-II and 27% cases (151 out of 540) for MOEA/D; there is a 9% tie (44 cases) for NSGA-II and 7% tie (39 cases) for MOEA/D. Particularly, in the cases where Pareto search wins, 377 (for NSGA-II) and 330 (for MOEA/D) of them come with statistical significance and a large effect size. In contrast, on the 75 (for NSGA-II) and 151 (for MOEA/D) cases where Pareto search loses, only 38 (for NSGA-II) and 104 (for MOEA/D) of them exhibit statistical significance



(a) # cases per result over 540 of the two objective cases

(b) # cases per result over 64 of the three objective cases

Fig. 4. The count on the pairwise comparison results between Pareto search and the best weighted search (W_{best} for each case, as identified from **RQ1**) over (a) 540 cases with 2 objectives (30 systems/projects × 9 weight vectors × 2 types of budget) and (b) 64 cases with 3 objectives (8 systems/projects × 4 weight vectors × 2 types of budget). The Pareto search with a win, lose, and tie refer to the case of $\hat{A}_{12} > 0.5$, $\hat{A}_{12} < 0.5$, and $\hat{A}_{12} = 0.5$, respectively. When there is a $\hat{A}_{12} \geq 0.56$ (or $\hat{A}_{12} \leq 0.44$) and $p < 0.05$, we say that the result is statistically significant.

and non-trivial effect size. The above is also consistent for the three objective case, where the Pareto search wins on 53% cases (34 out of 64) for NSGA-II and 62% (40 out of 64) for MOEA/D, on the majority of which are statistically significant. The improvement over the weighted search degrades slightly though.

To confirm whether the above observation applied to the systems/projects and SBSE problems, Tables 9 and 10 show the detailed results for two and three objective case, respectively. Here, we see that the overall conclusion remains unchanged for both numbers of the objectives: Pareto search often wins with a reasonably good degree at the 25th, 50th, and 75th percentiles for a majority of the systems/projects (up to 1.77× median improvement). Another observation is that the results are consistent across the different SBSE problems and their systems/projects, which further confirms the generality of the conclusion.

This is a rather surprising outcome as weighted search is guided by the exact weight vector used in the comparison, and hence it was generally believed to be better by the SBSE community. The result confirms our hypothesis and even reveals the significant superiority of Pareto search under clear preferences for **RQ2**.

> ***Finding:*** *Pareto search is significantly better for up to 77% cases of the multi-objective SBSE cases under reasonable convergence.*

5.2.3 Implication. Our findings for **RQ2** demonstrate that the *weighted search first* belief is problematic, as Pareto search can perform considerably better in general as long as there is a sufficient search budget. Recall that from Table 6, we note that such a budget differ significantly depending on the systems/projects and SBSE problems in hand, but essentially they do not have to be unrealistically high: it could well be a few hundreds of evaluations or in the magnitude of seconds.

Of course, whether the quality of solution or the resource required is more important is subject to the stakeholders' preferences and requirements; we can, however, suggest the following to the SBSE practitioners:

Table 9. Pairwise comparisons (by $\hat{A}_{12}$ and Wilcoxon rank-sum test) and the weighted scores for Pareto and weighted search on two objectives SBSE problems. ⊢●⊣ denotes the 25th, 50th, and 75th percentile of the weighted score. A pair summarizes 18 cases (9 weight vectors × 2 types of budget) on 100 runs each. W_{best} represents the best single objective algorithm and the normalization for the weighted search. $\mathcal{W}$ and $\mathcal{L}$ show how many cases the Pareto search wins ($\hat{A}_{12} > 0.5$) and loses ($\hat{A}_{12} < 0.5$) to the weighted counterpart, respectively. $\mathcal{T}$ denotes tie ($\hat{A}_{12} = 0.5$). The comparisons where the Pareto search and its weighed counterpart wins more are shown in blue and red, respectively. The number in the bracket shows how many win/lose is statistically significant, i.e., $\hat{A}_{12} \geq 0.56$ (or $\hat{A}_{12} \leq 0.44$) and $p < 0.05$. The one that with more statistically significant wins is highlighted in bold.

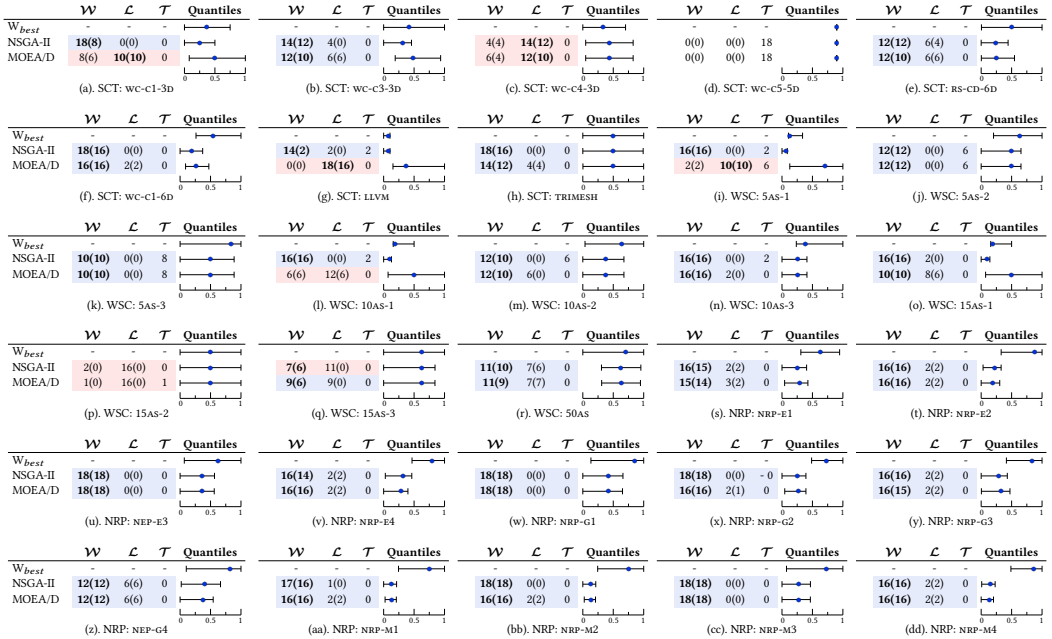
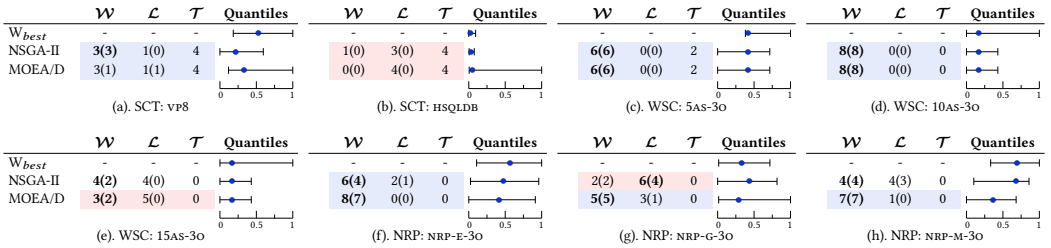
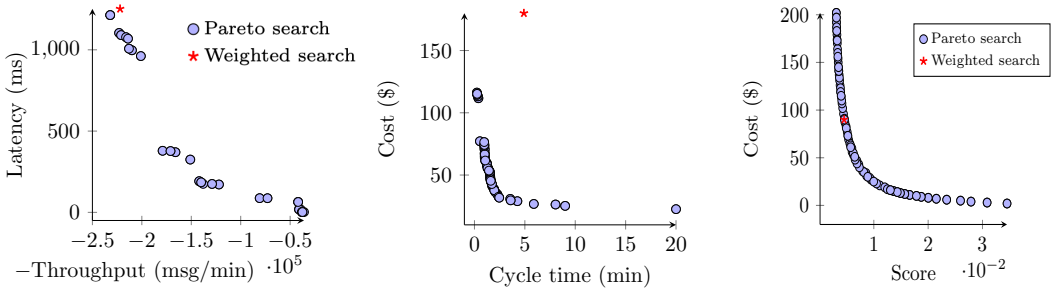


Table 10. Pairwise comparisons (by $\hat{A}_{12}$ and Wilcoxon rank-sum test) and the weighted scores for Pareto and weighted search on three objectives SBSE problems. A pair summarizes 8 cases (4 weight vectors × 2 types of budget) on 100 runs each. Formats are the same as Table 9.



Suggestion: *When the quality of the solution is a primary concern for the multi-objective SBSE problem in hand, use Pareto search by default even if there are clear preferences (weights between the objectives).*



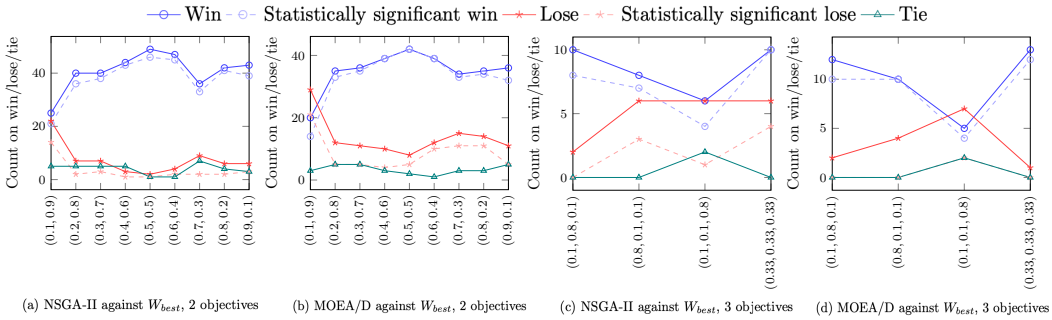
(a) RS-C4-6D, (0.5, 0.5) (b) 15-AS1, (0.4, 0.6) (c) NRP-M3, (0.7, 0.3)

Fig. 5. The left (a) and the middle (b) are examples where the result of weighted search is dominated by that of Pareto search; the right (c) is the example wherein a non-dominated solution has been obtained by weighted search, but the weighted fitness is worse than that of Pareto search.

5.2.4 Reason. The above results confirm the theoretical possibility discussed in Section 2.3. The diversity of solutions maintained by Pareto search helps to find the global optimum. In contrast, fine-grained comparability of the scalar fitness in weighted search may be easy to get stuck in local optima. especially for SBSE problems wherein the search space is large and/or with many complex local optimal regions. Figure 5(a) and (b) give the solutions found by Pareto search and weighted search in a typical run on two SCT and WSC cases. As can be seen, weighted search not only fails to find the global optimum but its solution is actually dominated by some solutions of Pareto search. This implies that weighted search may stagnate very easily during the search and the solution found is a local optimum.

Another reason of weighted search performs poorly is that, as discussed in Section 2.3, the stakeholders' preferences between the objectives may not be well reflected during the search since the weights are determined on the basis of the real bounds of the problem, whereas the normalization is done on the basis of the bound found during the search. This has been evident in Figure 5(c) — despite the fact that the solution found by weighted search is on the Pareto front, it can differ from the one that the stakeholders are really interested in, i.e., the weight vector $(0.7, 0.3)$.

In addition, it is worth mentioning that in a small number of cases (e.g., WC-C4-3C for SCT and 5AS-1 for WSC) Pareto search performs significantly worse than weighted search. The reason for this is due to the "spread" search manner of Pareto search. Pareto search maintains a well-diversified population, searching in parallel towards a number of locations (i.e., diverse trade-offs over the Pareto front). Such a spread search manner can be somehow detrimental if the goal of the search is to locate one particular solution on the Pareto front. For example, the crossover operation, which typically operates on two solutions distant from each other in Pareto search, is less likely to generate good offspring along with one of their parents' directions [46]. In contrast, weighted search has the search focus of the specific direction through the crossover between similar parent solutions, and is more likely to generate promising solutions along that direction. Therefore, the weighted search may find better solutions when the search landscape is fairly easy (e.g., without many local optima) and the scale of different objectives as well as their Pareto front ranges is commensurable.



(a) NSGA-II against W_{best}, 2 objectives (b) MOEA/D against W_{best}, 2 objectives (c) NSGA-II against W_{best}, 3 objectives (d) MOEA/D against W_{best}, 3 objectives

Fig. 6. The count on the pairwise comparison results between Pareto search and the best weighted search (W_{best}) across the weight vectors. Formats are the same as Figure 4.

5.3 RQ3: Sensitivity to the Weight Vector

5.3.1 Method. Answering **RQ3** requires us to examine whether the general observations from **RQ2** would change when looking at each specific weight vector. We do that by following the same settings as discussed in Section 5.2, including the same W_{best} for each case.

To provide more meaningful and intuitive illustrations, we report on the percentage gain of one's weighted score over the other:

$$\%\ \text{Gain} = \frac{1}{n} \times \sum_{i=1}^{n} \frac{y_i - x_i}{y_i} \times 100 \tag{9}$$

whereby x_i and y_i are the weighted score at the ith run for the Pareto search and W_{best}, respectively, in which the results of different runs are sorted in ascending order. n is the total number of runs (we have $n = 100$ in this work). A negative gain means that the Pareto search is even worse off.

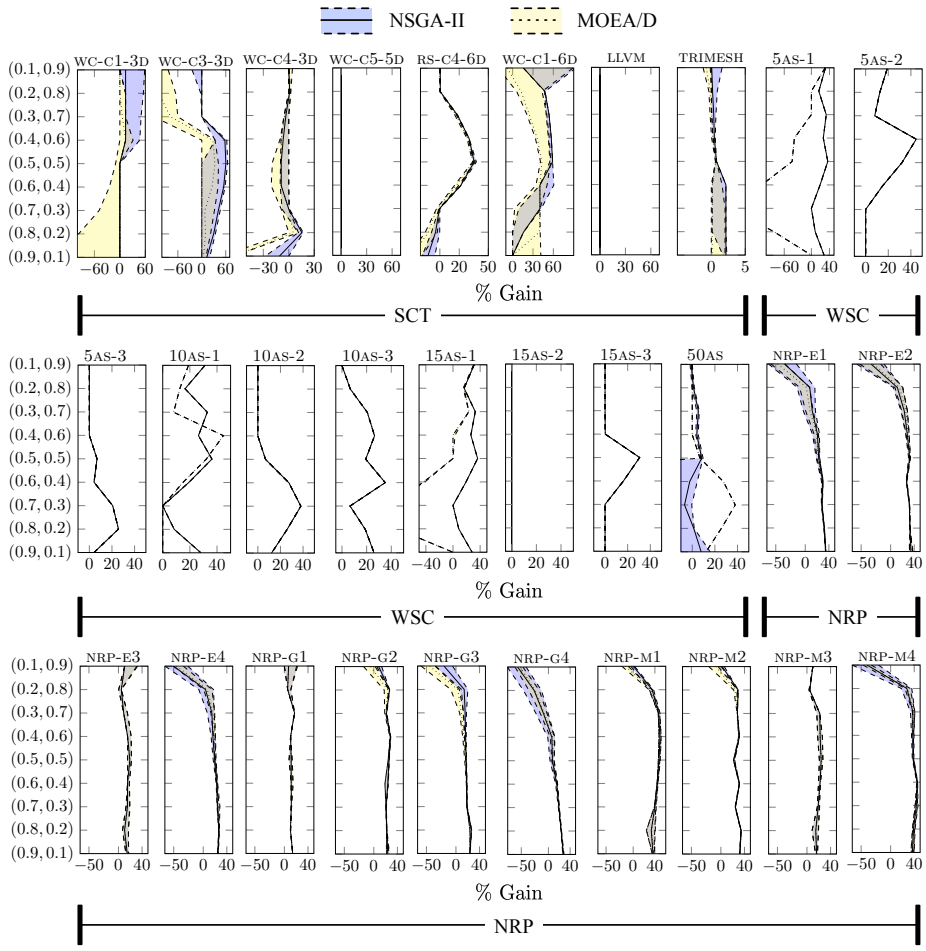
5.3.2 Result. From Figure 6, we see clearly that the gains achieved by Pareto search (on both NSGA-II and MOEA/D) over its weighted counterpart do fluctuate depending on the weight vector. However, Pareto search remains to win more in general. In particular, the majority of the wins by Pareto search are statistically significant with non-trivial effect size.

To better understand the results with respect to the systems/projects and SBSE problems, in Figure 7, we can obtain similar observations with considerably high percentage gain in general. The only case when Pareto search often loses is for NRP under some extreme weight vectors, e.g., $(0.1, 0.9)$. We also note that, for both NSGA-II and MOEA/D, the patterns of percentage gains across different systems/projects are more consistent in NRP when compared with SCT and WSC. This makes sense since the systems in the latter two SBSE problems can have a much more radically different nature compared with the project/versions in the former. This is rather clear with SCT where the variations are usually higher.

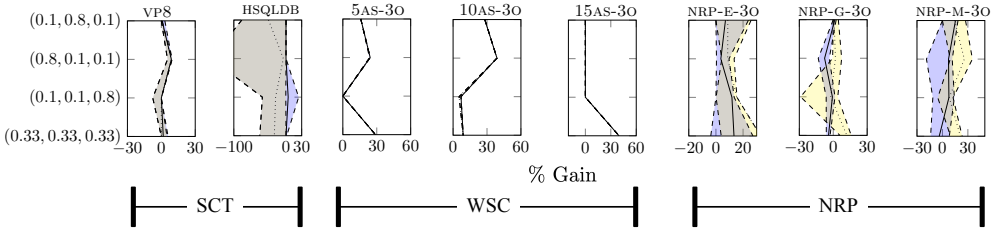
The above observations can be seen for both the two and three objective cases.

> ***Finding:*** *The extent to which Pareto search outperforms its weighted counterpart vary depending on the weight vector in the multi-objective SBSE cases. However, the overall result remains the same as we concluded from* ***RQ2****.*

5.3.3 Implication. We observe from the results that, for each system/project, the best gain achieved by Pareto search is generally centered at a particular range of the weight vectors, mostly around the middle range. For example, in the two objective cases, it can be between $(0.4, 0.6)$ and $(0.6, 0.4)$,



(a) 2 Objectives



(b) 3 Objectives

Fig. 7. Sensitivity of the 25th, 50th, and 75th percentile gain (%) of the Pareto search over the best weighted counterpart (W_{best}) to different weight vectors. Note that for three objective case, the middle range weight vector lies in the bottom.

for SCT and WSC; or from $(0.3, 0.7)$ to $(0.7, 0.3)$ for NRP. For three objective case, often the $(0.33, 0.33, 0.33)$ gives one of the best results. This suggests that different SBSE problems (and their systems/projects) possess different "comfort zones" on the given weights for Pareto search to outperform its weighted counterpart. However, we see clear patterns on when it is the most difficult for Pareto search to become more beneficial for a system/project, i.e., close to one (or both) extreme weights such as $(0.1, 0.9)$, $(0.9, 0.1)$, and $(0.1, 0.1, 0.8)$. Of course, this may be asymmetric: on the two objective case, Pareto search may have the lowest gain on $(0.1, 0.9)$ while achieving a reasonably well gain on $(0.9, 0.1)$ (see NRP), but it does bring it to our attention that one needs to be cautious when the given weight vector is close to the edge. As a result, we can make the following advice:

Suggestion: *In multi-objective SBSE, when the given weight vector is close to one extreme e.g.,* $(0.1, 0.9)$ *and* $(0.2, 0.8)$*, it is useful to further experimentally confirm the benefits of Pareto search in some preliminary runs.*

5.3.4 Reason. One reason that Pareto search under extreme weight vectors may not be as effective as under other weight vectors is that it can be hard for Pareto search to search for boundary solutions of the Pareto front. Compared to trade-off solutions (i.e., more central part on the Pareto front), on certain multi-objective problems, the boundary solutions can be very difficult to be found. They may be located in a region that is on the edge of search space, far away from the randomly generated initial population. Finding them needs the focus of the search toward their location. However, Pareto search maintains a well-diversified population, searching in parallel towards a number of locations (i.e., diverse trade-offs over the Pareto front). To be more specific, the crossover operation, which typically operates on two solutions distant from each other in Pareto search, is less likely to produce good offspring along with one of their parents' directions. In contrast, weighted search has the search focus of the specific direction through the crossover between similar parent solutions.

In addition, there is another reason, related to the search algorithm/optimizer itself, that Pareto search may not be that advantageous when the weights are closer to the extremes. Most multi-objective optimizers do not have a mechanism to preserve boundary solutions, such as MOEA/D [104] and NSGA-III [29]. That means that even if a boundary solution is generated during the search process, it may still be eliminated later. This is particularly true when the problem's Pareto front is convex (such as the WSC as shown in Figure 5b), where the boundary solutions can be seen "worse" than internal solutions in terms of convergence, thus directly being eliminated by the algorithm's selection criterion such as ϵ-dominance [57], grid ranking [103] criteria and shift-based density estimation criterion [65]. It is worth noting that this reason does not apply to the results obtained from NSGA-II since it has an explicit boundary solutions preservation mechanism and all nondominated solutions are incomparable in terms of convergence [28]. However, it may apply to other multi-objective optimizers such as MOEA/D. This is why, as shown in Figure 6a and Figure 6b, the MOEA/D tends to lose more to the weighted search compared with that of the NSGA-II when the weight vector is close to one extreme.

Lastly, it is worth mentioning that the fact that Pareto search may not work well in finding the boundary solutions is not alone in the SBSE area. Similar observations have been found on generic multi-objective problems in the evolutionary computation area [45, 99].

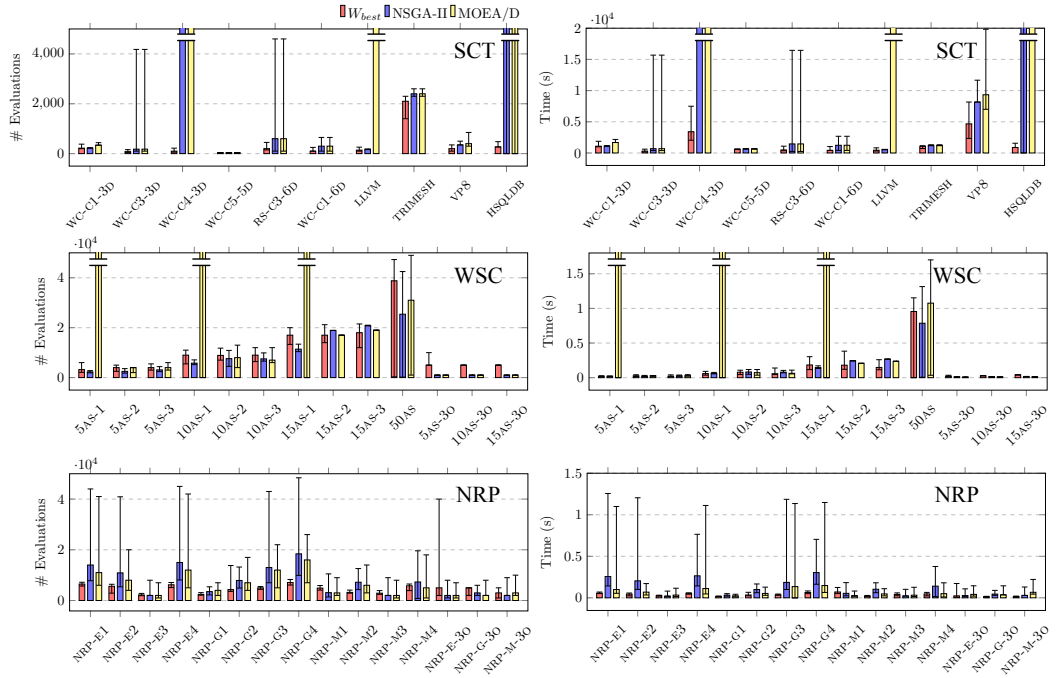


Fig. 8. The 25th, 50th, and 75th percentiles (across different weight vectors) of the resources consumed to reach (or outperform) the best median weighted score achieved by W_{best}. The broken bar denotes that the Pareto search cannot achieve the same results when the search budget is exhausted.

5.4 RQ4: Resource Efficiency

5.4.1 Method. To investigate **RQ4**, we use the same W_{best} for each case from **RQ2**. Specifically, in each case, we measure the resource efficiency in terms of the evaluations/time used to reach a certain level as:

(1) Identify a baseline, b, taken as the smallest amount of search budget (evaluations and time) that W_{best} consumes to achieve its best median result over 100 runs (says T).
(2) For each of the optimizers studied, find the smallest number of evaluations (and amount of time), m, at which the median weighted score (over 100 runs) is equivalent to or better than T.
(3) Report m and b across the different weight vectors for each system/project.

5.4.2 Result. As shown from Figure 8, we see that the Pareto search (both NSGA-II and MOEA/D) tends to be more resource-efficient on most system workflows in WSC, when using the number of evaluations as the search budget. However, the benefit remains unclear as the variations across different weight vectors remain high. In most of the remaining cases, the weighted search often consumes much less resources at the 25th, 50th, and 75th percentiles. This is particularly true for both SCT and NRP, where the same trends have been observed across the majority of the systems/projects and types of search budget.

The above observations still hold for both two and three objective cases.

Finding: *Weighted search is often more resource-efficient for multi-objective SBSE.*

5.4.3 Implication. The observation for **RQ4** is an interesting one: it reveals that weighted search does have its advantages over Pareto search; that is, it allows to converge to its best results with less resources. Yet, it is worth noting that, if we allow the search to continue, Pareto search would often reach a degree of quality that the best weighted counterpart would have never been able to achieve. However, the fact that weighed search is more resource-efficient in reaching its best is more preferable to some contexts under a limited search budget, meaning that the *weighted search first* belief is not entirely meaningless. We, therefore, suggest the following:

> ***Suggestion:*** *When the resource efficiency (search budget) is a more important factor for the multi-objective SBSE problem in hand, sticking with the existing belief to use weighted search.*

5.4.4 Reason. The reason that weighted search appears to be more efficient than Pareto search is easy to understand. Weighted search is conducted for the optimizer seeking the exclusive target (point) in the space based on the given weight vector, whereas Pareto search is conducted for the optimizer seeking the entire Pareto front, thus a waste of lots of recourse on solutions irrelevant to the weight vector.

In addition, a secondary reason for weighted search being less time consuming is that the fitness comparison between solutions in the population in weighted search requires $O(mn)$ computations (m is the number of objectives and n is the population size of the optimizer), less than that of NSGA-II in which the Pareto-based fitness comparison requires $O(mn^2)$ computations and that of MOEA/D in which the neighborhood-based fitness comparison requires $O(kmn)$ computations, where k is the neighborhood size.

6 A PRAGMATIC GUIDANCE

Drawing on the findings for our RQs, it is clear that a *"weighted search first"* belief under clear preferences can be harmful to SBSE. A more systematic justification is, therefore, required for making such an engineering decision. To that end, we codify pragmatic guidance that outlines the key processes.

As shown in Figure 9, the guidance starts by asking the stakeholders to feed a known weight vector into the first decision point (**D1**). Here, one can decide on whether the quality is more important than the resource taken, or vice versus. This is important, as different SBSE problems and contexts may impose different preferences.

If the resource efficiency (e.g., time or evaluations) is deemed as more important (choosing *resource efficiency* at **D1**), e.g., for SBSE problems like SCT where a single evaluation can be rather expensive, we recommend staying with the classic weighted search (based on **RQ4**). We can then choose an appropriate single-objective algorithm, based on theoretical or empirical understanding (Details of this can be found in [5, 41, 42], thus we do not cover this here). Next, in **D2**, we check whether the objectives need to be normalized for making them commensurable. If the objectives are commensurable in nature, we proceed to **D4**, and since we certainly prefer weighted search which performs reasonably better (than Pareto search) in terms of resource efficiency during the search, the process could end. Note that here, the search needs to be completed under a given search budget as resource consumption is more important.

The above is an ideal case, however, it is likely that most SBSE problems have objectives with radically different scales. Therefore, as revealed by **RQ1**, the normalization method can play an integral role in such a case. In **D3**, we ask if there are extra resources that allow for a thorough comparison between normalization methods. If there are not, we suggest using `Dynamic`, which has been shown to be generally safe in **RQ1**. Otherwise, experimental comparison among the methods (at least the four we consider in this work) in preliminary runs is desired (under a testing search

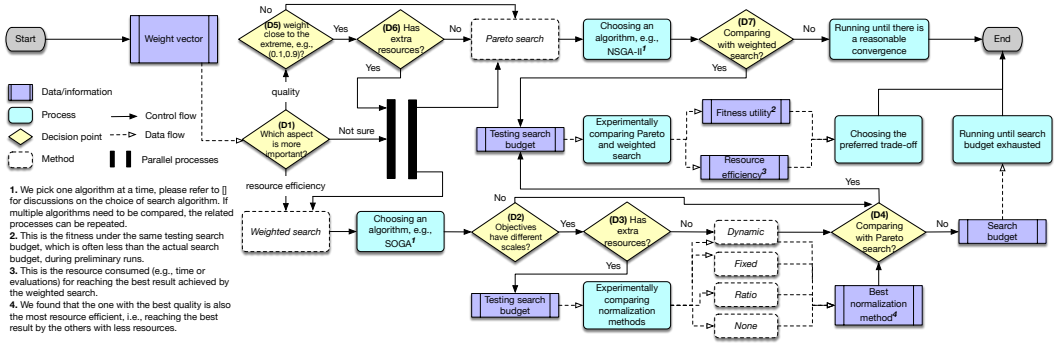


Fig. 9. A pragmatic guidance of how to choose between Pareto and weighted search under clear preference of weights in multi-objective SBSE.

budget[5]). Note that, at this point, `Ratio` can be ruled out from consideration to save more resources, since it is generally the worst compared with the other three. Here, there is no need to compare with the Pareto search.

In another situation, if the quality is much more preferred regardless of the resources needed, Pareto search can be a more ideal choice (based on **RQ2**), i.e., choosing *quality* at **D1**. This is not surprising, because in certain SBSE problems, such as NRP, satisfying the stakeholders with better (even slightly) fitness utility can often bring significantly more revenues [33]. One would also need to choose a search algorithm for Pareto search (see [27, 86]), although NSGA-II seems to be a more preferred choice according to various SBSE surveys [27, 41, 86]. Note that in this case, there is no fixed search budget given, instead one should allow the search to achieve reasonable convergence (e.g., the solutions do not change for certain generations), as the quality is more important. A particular step required here is that, in **D5** and **D6**, we ask whether the weight is close to one extreme and if extra resources are available, respectively. A *No* to either would proceed to the Pareto search directly. Yet, if there are positive answers to both **D5** and **D6**, additional studies are recommended. This is because, as we have shown in **RQ3**, the worst gains of Pareto search over the best weighted counterpart often occur under the weight vector like $(0.2, 0.8)$ and $(0.1, 0.9)$ (the other extreme is also applied). Therefore, to further ensure the benefit of Pareto search under such cases, additional experiments in preliminary runs to confirm the quality gains of Pareto search are desirable under a testing search budget (answering *Yes* to **D4** and **D7**, but only the fitness utility is important).

Indeed, albeit not always possible, experimentally comparing Pareto and weighted search on a case by case manner in preliminary runs can be helpful on their selection. Therefore, in such case the answer to **D4** and **D7** would be *Yes*. In fact, this is also the only way to go in the case that SBSE practitioners have absolutely no clue about whether the quality or resource efficiency is more important, i.e., choosing *not sure* at **D1**. In such a situation, as shown from the guidance, we suggest one to obtain two outcomes: (1) the quality of results when both Pareto and weighted search under a testing search budget and (2) the resource consumed by both in order to reach the best result achieved by weighted search. One can then pick the preferred trade-off with respect to the quality and resource consumed, as achieved by Pareto search and weighted search.

[5]This is only for the preliminary runs, which is often smaller than the actual budget for the ultimate run.

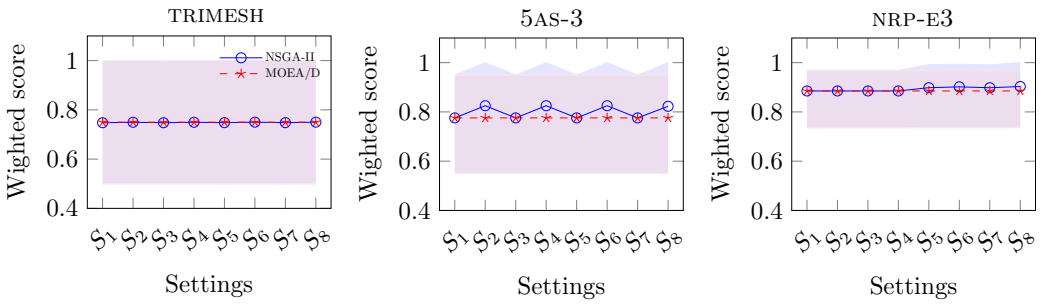


Fig. 10. The parameter sensitivity of Pareto search on the most obvious cases over 8 different settings of population size, crossover rate, and mutation rate. The line shows the median while the highlighted area indicates the 25th and 75th percentiles of the weighted score over 9 weight vectors and 100 runs each. S_1 denotes the setting we applied in this work.

7 DISCUSSION

In this section, we discuss a few important factors in our study.

7.1 Sensitivity of Parameters

While we have followed the parameter settings used by existing work, it is important to confirm the sensitivity of parameters to the results, particularly on the Pareto optimizers. To that end, for both NSGA-II and MOEA/D, we examine some common settings of the parameters: mutation rate of $\{0.01, 0.1\}$, crossover rate of $\{0.8, 0.9\}$, and population size of $\{X, \frac{X}{2}\}$ (where X is the size used in Table 6, depending on the problem), leading to a total of 8 combinatorial settings.

We found that for each SBSE problem, the trends are consistent across the systems/projects (regardless of the number of objectives), hence in Figure 10, we plot the most obvious case from each problem (i.e., the one with the highest deviation across the settings). From this, we obtained two observations:

- The resulted weighted score across different settings do not vary much.
- The setting we applied for the **RQs** (i.e., S_1) is one of the best among the others.

The above indicate that the conclusions drawn for the **RQs** are stable, as a reasonable change of the parameter setting leads to very little impact on the result.

7.2 On > 3 Objectives

We have shown that our conclusions are valid for both two and three objective cases in multi-objective SBSE. Yet, certain SBSE problems could consider more than three objectives, i.e., what has been known as many-objective search/optimization problems [47]. Many-objective optimization poses big challenges to any algorithms aiming to find/approximate the whole Pareto front of the problem.

We anticipate that the benefits of Pareto search would be blurred in such a many-objective setting, as what we have shown in Section 5, especially for certain optimizers such as NSGA-II. In fact, it has been well studied in the evolutionary computation community that the performance of many well-established multi-objective evolutionary algorithms falls rapidly with the increase of the number of objectives, particularly for Pareto-based search algorithms (e.g., NSGA-II) which only use the Pareto dominance relation to distinguish between solutions with respect to their

convergence [66, 96]. Recent studies even show that mainstream algorithms like NSGA-II, MOEA/D, and IBEA even completely fail on some four-objective problems [64].

In contrast, increasing search space has much less effect on weighted search since it aggregates the objectives into a scalar value by a weight vector and aims to locate a point of the Pareto front. However, a downside of using weighted search in the many-objective setting is that the higher the objective dimension the more difficult it is for the stakeholders to specify a weight vector a priori [40].

7.3 Threats to Validity

To ensure **construct validity**, we use the weighted sum of a given weight vector as the sole metric, which matches precisely with the need to verify our hypothesis. To mitigate threats caused by the stochastic nature of the optimizers, we repeat 100 runs for each case, with validation from Wilcoxon rank-sum test, $\hat{A}_{12}$ effect size, and Scott-Knott test, as commonly recommended for SBSE [4, 51, 75]. To ensure the strongest statistical power, we conduct pairwise comparisons in our study.

Two aspects may form threats to **internal validity** in our study:

- **Optimizer setting:** In this work, we follow what has been shown to be effective for a SBSE problem in the literature, as our aim is to compare the most common practices. The only part we could not have found for sure is the search budget, which is highly problem-dependent. To tackle such, we have followed carefully designed criteria (Section 4.2), including both evaluation and time budget, that strike for a balance between reasonable convergence and the time required. We have found that the parameter settings tend to be appropriate and the sensitivity of Pareto search to the parameters are marginal, as discussed in Section 7.1.
- **Weight vector:** We used the most common weight vector for a pragmatic reason. That is, in the two objective cases, we choose nine key vectors that are evenly spread across the space; for three objective cases, we used the edge and middle vectors, e.g., $(0.1, 0.1, 0.8)$ and $(0.33, 0.33, 0.33)$. Indeed, this list cannot cover all the possible scenarios, but they are good representatives of the most likely cases.

Threats to **external validity** can come from various sources, including:

- **SBSE problem:** In this work, we select the most representative SBSE problems from several surveys [27, 41, 63, 86] based on carefully codified rules (Section 3). Indeed, this list of the studied problem is not exhaustive, and we did not consider some popular problems, such as TCG (due to the nonmonotonic relation between objective and evaluation metric) and automatic refactoring [72] (which is the 6th most popular one from the surveys, but we limited to the top 5 due to resource constraint). We hope our work serves as a first step to open a dialogue on this important topic for the SBSE community, based on which future work can extend this study to cover the SBSE problems that we omitted.
- **Optimizer:** In this study, four widely used optimizers for the weighted search are examined, together with four different normalization methods, which are concluded from well-known SBSE surveys. For the Pareto search, we choose NSGA-II as the representative of the Pareto search due mainly to its prevalence and similarity in terms of algorithmic design to the weighted counterpart. We also examine MOEA/D because it uses multiple weights vector to reveal the Pareto front — a similar property in weighted search. We acknowledge that different optimizers may have diverse "comfort zones" for a given SBSE problem, and an extended study may be required for future work.
- **Number of objectives:** Our study covers two and three objective cases in SBSE, hence the results may not be generilizable to higher dimension cases of the objectives. However, two or three objectives are the most common studied problem for SBSE, as summarized by Sayyad

and Ammar [86]. Further, as we discussed in 7.2, there are known studies that confirm some Pareto optimizers can be severely affected by the number of objectives, e.g., NSGA-II.

- **SBSE constraints:** We do not consider any constraint for the three multi-objective SBSE problems, which have also been studied in other SBSE work [20]. For example, in SCT, a configuration option cannot be used unless another has been turned on. Indeed, those constraints, when considered, may affect the results on both Pareto and weighted search, as they would inevitably complicate the problem and potentially make the global optimum even more difficult to find. Since this study is the first comprehensive work to compare Pareto and weighted search for SBSE, we started from the simplest assumption where the constraints are omitted. Some of our findings are still exciting, for example, we have shown that, even with such a simpler case, the weighted search has been outperformed by Pareto search in terms of the exact weights that guide it. This can then serve as a foundation for future work to consider a more abnormal landscape of the SBSE problems, i.e., by having complex constraints.

We would like to stress that, in this work, we do not aim to exhaustively verify our hypothesis across all situations but to examine whether it is the case under the representative scenarios of SBSE. Nonetheless, we do agree that additional replication studies that extend all (or some) of the above aspects may prove fruitful.

8 RELATED WORK

Here we review the prior work for multi-objective SBSE in relation to the purpose of this work.

8.1 Multi-Objective SBSE with Weighted Search

Indeed, it is not uncommon to assume clear weights for different objectives under multi-objective SBSE, such as software configuration tuning [7, 84, 90], web service composition [9, 95], next release planning [33], software project scheduling [10], and software modularization [44]; in fact, this is occasionally referred to as an advantage rather than a limitation. For example, Bowers et al. [7] and Shahbazian et al. [90] argue that, in software configuration tuning, being able to specify weights provides more flexibility for the stakeholder to freely set preferences depending on the context.

Existing work applies various normalization methods for the weighted search in multi-objective SBSE when the objectives do not naturally commensurable. For example, Shahbazian et al. [90] dynamically update the weights during search such that the different performance objectives are rescaled when tuning software performance. In test case generation, Wang et al. [98] and Pradhan et al. [80] use $\frac{v}{1+v}$ to normalize an objective's value v, yet this method only converts the values into $[0, 1]$ without standardizing them, and hence the fitness can still be dominated by the objectives with relatively larger magnitude (e.g., cost over coverage); this, as we have shown in Section 5, tends to severely affect the result for the SBSE problems studied.

8.2 Multi-Objective SBSE with Pareto Search

In multi-objective SBSE problems, Pareto search is often regarded as a better strategy when it is impossible to clearly quantify the weights, or it is desirable for the stakeholders to examine the whole Pareto front. Indeed, this is often the case when the number of objectives is high (e.g., three or more) and it has been becoming the standard for certain SBSE problems, such as the software product line engineering [69, 79, 87, 88] and code refactoring [43, 71, 76].

8.3 Comparison on Pareto and Weighted Search

The first wave of work that compares Pareto and weighted search in multi-objective SBSE appeared more than a decade ago [43, 56, 105] till more recently [102], each of which studies a different SBSE problem, such as next release planning and test case generation. By plotting the result on each objective, the above work demonstrates an obvious but perhaps "new result" in SBSE by that time: the Pareto search provides better insights on the trade-off surface as it approximates the Pareto front. More recently, Wang et al. [98] and Pradhan et al. [80] conduct empirical studies that compare weighted search (i.e., *FW* in their work[6]) with Pareto search using hypervolume (HV) over a few multi-objective SBSE problems, based on which they unsurprisingly concluded that the Pareto search is better as it has higher HV value.

Those studies, albeit offering interesting findings, are unfair when comparing Pareto and weighted search. This is because they overlook the fact that only a particular solution is of interest to the stakeholders in the presence of clear preferences (rather than the whole Pareto front). As a result, interpreting each objective individually from the Pareto front approximation cannot well respect the given preferences. Likewise, HV is not suitable since it measures how well the solution set approximates the Pareto front [109]. Another unfairness raised from the fact that the time budget has not been considered, which, as we have shown, could be consumed quite differently even with the same number of evaluations. Our study addresses all of the above issues from prior work.

Mkaouer et al. [76] attempt to achieve a more fair comparison on the software refactoring problem by contrasting the knee solution from the Pareto search to the solution obtained by an equally-weighted search. They concluded that the Pareto search is better, as it leads to a better average of the objectives' values. However, the knee solution may not be fully in line with the solution obtained by equally weighted search (as it can be most fitted by some other weight vector). A direct comparison between them may not be well justified.

A recent study by Alizadeh et al. [1] seeks to compare both strategies by allowing developers to qualitatively evaluate the solution(s). Since the Pareto search favors many solutions at each run, as expected, the developer concluded that the results of the weighted search are closer to their preferences since there is less "cognition noise" involved. While this may be the most direct way to assess their usefulness, the evaluation can be, however, biased by human judgment.

The safest option is probably to quantitatively compare them via the weight vector that is used to guide the weighted-search, as what has been done by Praditwong et al. [81] and this work. Yet, unlike our work, Praditwong et al. [81] study the software modularization problem by comparing NSGA-II with hill-climbing—two fundamentally different optimizers. However, their work differs from ours in the sense that (1) it is questionable that whether the simple hill-climbing can be a good representative of the weighted search; (2) only one SBSE problem is studied; (3) the best normalization method has not been justifiably chosen; and (4) an identical time budget has not been considered.

Overall, through extensive experiments, our work differs from prior studies in that we provide the explicit answer, explanation, and insights to an unexplored question: given clear preferences, identical search budget, can Pareto search converge to the same or better result than weighted search in terms of a given weight vector?

9 CONCLUSION

In this paper, we systematically compare Pareto search and weighted search under clear preferences on a large scale empirical study with 604 cases, including 38 systems/projects from three representative multi-objective SBSE problems, different weight vectors, and two search budget

[6]This is because *FW* is the only one that assumes clear preferences without approximating the Pareto front.

types. Our key finding challenges the existing *weighted search first* belief: we show that, although the weighted search is more resource-efficient for reaching certain levels of the result, Pareto search is most of the time (up to 77%) significantly better under a sufficient search budget. In particular, the more search budget required by Pareto search may be practically trivial, e.g., in a magnitude of seconds or less. Drawing on the findings, we suggest for the first time the following suggestions to the practitioners of multi-objective SBSE:

- When the quality of the solution is a primary concern for the multi-objective SBSE problem in hand, using Pareto search by default even if there are clear preferences.
- When the specified weight vector is close to one extreme e.g. $(0.1, 0.9)$ and $(0.2, 0.8)$, it is desirable to further experimentally confirm the benefits of Pareto search in preliminary runs.
- When the resource efficiency (search budget) is a more important factor for the multi-objective SBSE problem in hand, sticking with the existing belief to use weighted search, yet it is non-trivial to decide on how the objectives need to be normalized when necessary.
- For weighted search, experimentally comparing the normalization methods (at least the four in this work) whenever the conditions permitted; the `Ratio` can be omitted in case the resource is limited. Otherwise, using `Dynamic` by default.

We codify the above as pragmatic guidance, hoping to provide a clear view on the choice between Pareto search and weighted search for the community. Several future opportunities can be derived from this work to advance the understanding of the topic, such as extending the study to wider range of SBSE problems and scenarios; replacing the weighted sum with other scalarizing functions (e.g., Tchebycheff) in weighted search; and linking the findings to the shape of the SBSE problem's Pareto front.

REFERENCES

[1] Vahid Alizadeh, Houcem Fehri, and Marouane Kessentini. 2019. Less is More: From Multi-objective to Mono-objective Refactoring via Developer's Knowledge Extraction. In *19th International Working Conference on Source Code Analysis and Manipulation, SCAM 2019, Cleveland, OH, USA, September 30 - October 1, 2019*. IEEE, 181–192.

[2] Giuliano Antoniol, Massimiliano Di Penta, and Mark Harman. 2005. Search-Based Techniques Applied to Optimization of Project Planning for a Massive Maintenance Project. In *21st IEEE International Conference on Software Maintenance (ICSM 2005), 25-30 September 2005, Budapest, Hungary*. IEEE Computer Society, 240–249.

[3] Allysson Allex Araújo and Matheus Paixão. 2014. Machine Learning for User Modeling in an Interactive Genetic Algorithm for the Next Release Problem. In *Search-Based Software Engineering - 6th International Symposium, SSBSE 2014, Fortaleza, Brazil, August 26-29, 2014. Proceedings (Lecture Notes in Computer Science, Vol. 8636)*, Claire Le Goues and Shin Yoo (Eds.). Springer, 228–233. https://doi.org/10.1007/978-3-319-09940-8_17

[4] Andrea Arcuri and Lionel C. Briand. 2011. A practical guide for using statistical tests to assess randomized algorithms in software engineering. In *Proceedings of the 33rd International Conference on Software Engineering, ICSE 2011, Waikiki, Honolulu , HI, USA, May 21-28, 2011*. 1–10.

[5] Anthony J. Bagnall, Victor J. Rayward-Smith, and Ian M. Whittley. 2001. The next release problem. *Inf. Softw. Technol.* 43, 14 (2001), 883–890. https://doi.org/10.1016/S0950-5849(01)00194-X

[6] Paul Baker, Mark Harman, Kathleen Steinhöfel, and Alexandros Skaliotis. 2006. Search Based Approaches to Component Selection and Prioritization for the Next Release Problem. In *22nd IEEE International Conference on Software Maintenance (ICSM 2006), 24-27 September 2006, Philadelphia, Pennsylvania, USA*. IEEE Computer Society, 176–185. https://doi.org/10.1109/ICSM.2006.56

[7] Kate M. Bowers, Erik M. Fredericks, and Betty H. C. Cheng. 2018. Automated Optimization of Weighted Non-functional Objectives in Self-adaptive Systems. In *Search-Based Software Engineering - 10th International Symposium, SSBSE 2018, Montpellier, France, September 8-9, 2018, Proceedings (Lecture Notes in Computer Science, Vol. 11036)*, Thelma Elita Colanzi and Phil McMinn (Eds.). Springer, 182–197.

[8] Radu Calinescu, Milan Ceska Jr., Simos Gerasimou, Marta Kwiatkowska, and Nicola Paoletti. 2018. Efficient synthesis of robust models for stochastic systems. *Journal of Systems and Software* 143 (2018), 140–158.

[9] Gerardo Canfora, Massimiliano Di Penta, Raffaele Esposito, and Maria Luisa Villani. 2005. An approach for QoS-aware service composition based on genetic algorithms. In *Genetic and Evolutionary Computation Conference, GECCO 2005, Proceedings, Washington DC, USA, June 25-29, 2005*, Hans-Georg Beyer and Una-May O'Reilly (Eds.). ACM, 1069–1075.

[10] Carl K. Chang, Hsinyi Jiang, Yu Di, Dan Zhu, and Yujia Ge. 2008. Time-line based model for software project scheduling with genetic algorithms. *Inf. Softw. Technol.* 50, 11 (2008), 1142–1154. https://doi.org/10.1016/j.infsof.2008.03.002

[11] Tao Chen. 2019. 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*, Marin Litoiu, Siobhán Clarke, and Kenji Tei (Eds.). ACM, 157–168. https://doi.org/10.1109/SEAMS.2019.00029

[12] Tao Chen. 2022. Lifelong dynamic optimization for self-adaptive systems: fact or fiction?. In *SANER '22: 29th IEEE International Conference on Software Analysis, Evolution and Reengineering, Hawaii, United States, March 15-18 2022*. IEEE.

[13] Tao Chen and Rami Bahsoon. 2014. Symbiotic and sensitivity-aware architecture for globally-optimal benefit in self-adaptive cloud. In *9th International Symposium on Software Engineering for Adaptive and Self-Managing Systems, SEAMS 2014, Proceedings, Hyderabad, India, June 2-3, 2014*, Gregor Engels and Nelly Bencomo (Eds.). ACM, 85–94. https://doi.org/10.1145/2593929.2593931

[14] Tao Chen and Rami Bahsoon. 2015. Toward a Smarter Cloud: Self-Aware Autoscaling of Cloud Configurations and Resources. *Computer* 48, 9 (2015), 93–96. https://doi.org/10.1109/MC.2015.278

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

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

[17] Tao Chen, Rami Bahsoon, and Georgios Theodoropoulos. 2013. Dynamic QoS Optimization Architecture for Cloud-Based DDDAS. In *Proceedings of the International Conference on Computational Science, ICCS 2013, Barcelona, Spain, 5-7 June, 2013 (Procedia Computer Science, Vol. 18)*, Vassil N. Alexandrov, Michael Lees, Valeria V. Krzhizhanovskaya, Jack J. Dongarra, and Peter M. A. Sloot (Eds.). Elsevier, 1881–1890. https://doi.org/10.1016/j.procs.2013.05.357

[18] Tao Chen, Rami Bahsoon, and Xin Yao. 2018. A Survey and Taxonomy of Self-Aware and Self-Adaptive Cloud Autoscaling Systems. *ACM Comput. Surv.* 51, 3 (2018), 61:1–61:40. https://doi.org/10.1145/3190507

[19] Tao Chen, Rami Bahsoon, and Xin Yao. 2020. Synergizing Domain Expertise With Self-Awareness in Software Systems: A Patternized Architecture Guideline. *Proc. IEEE* 108, 7 (2020), 1094–1126. https://doi.org/10.1109/JPROC.2020.2985293

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

[21] Tao Chen and Miqing Li. 2021. MMO: Meta Multi-Objectivization for Software Configuration Tuning. *CoRR* abs/2112.07303 (2021). arXiv:2112.07303 https://arxiv.org/abs/2112.07303

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

[23] Tao Chen, Miqing Li, Ke Li, and Kalyanmoy Deb. 2020. Search-Based Software Engineering for Self-Adaptive Systems: Survey, Disappointments, Suggestions and Opportunities. *CoRR* abs/2001.08236 (2020).

[24] Tao Chen, Miqing Li, and Xin Yao. 2018. On the effects of seeding strategies: a case for search-based multi-objective service composition. In *Proceedings of the Genetic and Evolutionary Computation Conference*. ACM, 1419–1426.

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

[26] Shang-Wen Cheng, David Garlan, and Bradley R. Schmerl. 2006. Architecture-based self-adaptation in the presence of multiple objectives. In *Proceedings of the 2006 international workshop on Self-adaptation and self-managing systems, SEAMS 2006, Shanghai, China, May 21-22, 2006*, Betty H. C. Cheng, Rogério de Lemos, Stephen Fickas, David Garlan, Jeff Magee, Hausi A. Müller, and Richard Taylor (Eds.). ACM, 2–8.

[27] Thelma Elita Colanzi, Wesley K. G. Assunção, Silvia R. Vergilio, Paulo Roberto Farah, and Giovani Guizzo. 2020. The Symposium on Search-Based Software Engineering: Past, Present and Future. *Inf. Softw. Technol.* 127 (2020), 106372.

[28] Kalyanmoy Deb, Samir Agrawal, Amrit Pratap, and T. Meyarivan. 2002. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Trans. Evol. Comput.* 6, 2 (2002), 182–197. https://doi.org/10.1109/4235.996017

[29] Kalyanmoy Deb and Himanshu Jain. 2014. An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach, part I: Solving problems with box constraints. *IEEE Transactions on Evolutionary Computation* 18, 4 (2014), 577–601.

[30] Yichuan Ding, Sandra Gregov, Oleg Grodzevich, Itamar Halevy, Zanin Kavazovic, Oleksandr Romanko, Tamar Seeman, Romy Shioda, and Fabien Youbissi. 2006. Discussions on normalization and other topics in multi-objective optimization. In *Proceedings to the Fields-MITACS Industrial Problem Solving Workshop, Toronto*.

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

[32] Michael TM Emmerich and André H Deutz. 2018. A tutorial on multiobjective optimization: fundamentals and evolutionary methods. *Natural computing* 17, 3 (2018), 585–609.

[33] Martin S. Feather and Tim Menzies. 2002. Converging on the Optimal Attainment of Requirements. In *10th Anniversary IEEE Joint International Conference on Requirements Engineering (RE 2002), 9-13 September 2002, Essen, Germany*. IEEE Computer Society, 263–272. https://doi.org/10.1109/ICRE.2002.1048537

[34] Gregory Gay, Matt Staats, Michael W. Whalen, and Mats Per Erik Heimdahl. 2014. Moving the goalposts: coverage satisfaction is not enough. In *7th International Workshop on Search-Based Software Testing, SBST 2014, Hyderabad, India, June 2, 2014*, Phil McMinn and Mark Harman (Eds.). ACM, 19–22. https://doi.org/10.1145/2593833.2593837

[35] Jiangyi Geng, Shi Ying, Xiangyang Jia, Ting Zhang, Xuan Liu, Lanqing Guo, and Jifeng Xuan. 2018. Supporting Many-Objective Software Requirements Decision: An Exploratory Study on the Next Release Problem. *IEEE Access* 6 (2018), 60547–60558.

[36] Simos Gerasimou, Radu Calinescu, and Giordano Tamburrelli. 2018. Synthesis of probabilistic models for quality-of-service software engineering. *Autom. Softw. Eng.* 25, 4 (2018), 785–831.

[37] David E Goldberg. 2006. *Genetic algorithms*. Pearson Education India.

[38] Vincent Granville, Mirko Krivánek, and Jean-Paul Rasson. 1994. Simulated Annealing: A Proof of Convergence. *IEEE Trans. Pattern Anal. Mach. Intell.* 16, 6 (1994), 652–656.

[39] Mark Harman. 2007. 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*, Lionel C. Briand and Alexander L. Wolf (Eds.). IEEE Computer Society, 342–357. https://doi.org/10.1109/FOSE.2007.29

[40] Mark Harman. 2010. The relationship between search based software engineering and predictive modeling. In *Proceedings of the 6th International Conference on Predictive Models in Software Engineering, PROMISE 2010, Timisoara, Romania, September 12-13, 2010*, Tim Menzies and Günes Koru (Eds.). ACM, 1. https://doi.org/10.1145/1868328.1868330

[41] Mark Harman, S Afshin Mansouri, and Yuanyuan Zhang. 2012. Search-based software engineering: Trends, techniques and applications. *ACM Computing Surveys (CSUR)* 45, 1 (2012), 11.

[42] Mark Harman and Phil McMinn. 2010. A Theoretical and Empirical Study of Search-Based Testing: Local, Global, and Hybrid Search. *IEEE Trans. Software Eng.* 36, 2 (2010), 226–247. https://doi.org/10.1109/TSE.2009.71

[43] Mark Harman and Laurence Tratt. 2007. Pareto optimal search based refactoring at the design level. In *Genetic and Evolutionary Computation Conference, GECCO 2007, Proceedings, London, England, UK, July 7-11, 2007*, Hod Lipson (Ed.). ACM, 1106–1113.

[44] Jinhuang Huang and Jing Liu. 2016. A similarity-based modularization quality measure for software module clustering problems. *Inf. Sci.* 342 (2016), 96–110. https://doi.org/10.1016/j.ins.2016.01.030

[45] Hisao Ishibuchi and Yusuke Nojima. 2007. Optimization of scalarizing functions through evolutionary multiobjective optimization. In *International Conference on Evolutionary Multi-Criterion Optimization*. Springer, 51–65.

[46] Hisao Ishibuchi and Youhei Shibata. 2003. A similarity-based mating scheme for evolutionary multiobjective optimization. In *Genetic and Evolutionary Computation Conference*. Springer, 1065–1076.

[47] Hisao Ishibuchi, Noritaka Tsukamoto, and Yusuke Nojima. 2008. Evolutionary many-objective optimization: A short review. In *Proceedings of the IEEE Congress on Evolutionary Computation, CEC 2008, June 1-6, 2008, Hong Kong, China*. IEEE, 2419–2426. https://doi.org/10.1109/CEC.2008.4631121

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

[49] Chandrashekar Jatoth, G. R. Gangadharan, and Rajkumar Buyya. 2017. Computational Intelligence Based QoS-Aware Web Service Composition: A Systematic Literature Review. *IEEE Trans. Serv. Comput.* 10, 3 (2017), 475–492.

[50] Prashant Kadam and Supriya Bhalerao. 2010. Sample size calculation. *International journal of Ayurveda research* 1, 1 (2010), 55.

[51] Vigdis By Kampenes, Tore Dybå, Jo Erskine Hannay, and Dag I. K. Sjøberg. 2007. A systematic review of effect size in software engineering experiments. *Information & Software Technology* 49, 11-12 (2007), 1073–1086.

[52] Adrian Klein, Fuyuki Ishikawa, and Shinichi Honiden. 2011. Efficient Heuristic Approach with Improved Time Complexity for Qos-Aware Service Composition. In *IEEE International Conference on Web Services, ICWS 2011, Washington, DC, USA, July 4-9, 2011*. IEEE Computer Society, 436–443. https://doi.org/10.1109/ICWS.2011.60

[53] Satish Kumar, Rami Bahsoon, Tao Chen, and Rajkumar Buyya. 2019. Identifying and Estimating Technical Debt for Service Composition in SaaS Cloud. In *2019 IEEE International Conference on Web Services, ICWS 2019, Milan, Italy, July 8-13, 2019*, Elisa Bertino, Carl K. Chang, Peter Chen, Ernesto Damiani, Michael Goul, and Katsunori Oyama (Eds.).

IEEE, 121–125. https://doi.org/10.1109/ICWS.2019.00030

[54] Satish Kumar, Rami Bahsoon, Tao Chen, Ke Li, and Rajkumar Buyya. 2018. 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, 972–979. https://doi.org/10.1109/PADSW.2018.8644640

[55] Satish Kumar, Tao Chen, Rami Bahsoon, and Rajkumar Buyya. 2020. 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*, Shinichi Honiden, Elisabetta Di Nitto, and Radu Calinescu (Eds.). ACM, 96–107. https://doi.org/10.1145/3387939.3391604

[56] Kiran Lakhotia, Mark Harman, and Phil McMinn. 2007. A multi-objective approach to search-based test data generation. In *Genetic and Evolutionary Computation Conference, GECCO 2007, Proceedings, London, England, UK, July 7-11, 2007*, Hod Lipson (Ed.). ACM, 1098–1105.

[57] Marco Laumanns, Lothar Thiele, Kalyanmoy Deb, and Eckart Zitzler. 2002. Combining convergence and diversity in evolutionary multiobjective optimization. *Evolutionary Computation* 10, 3 (2002), 263–282.

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

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

[60] Lingbo Li, Mark Harman, Emmanuel Letier, and Yuanyuan Zhang. 2014. Robust next release problem:handling uncertainty during optimization. In *Conference on Genetic and Evolutionary Computation*. 1247–1254.

[61] Miqing Li. 2021. Is Our Archiving Reliable? Multiobjective Archiving Methods on "Simple" Artificial Input Sequences. *ACM Transactions on Evolutionary Learning and Optimization* 1, 3 (2021), 1–19.

[62] Miqing Li, Tao Chen, and Xin Yao. 2018. A Critical Review of "A Practical Guide to Select Quality Indicators for Assessing Pareto-Based Search Algorithms in Search-Based Software Engineering": Essay on Quality Indicator Selection for SBSE. In *2018 IEEE/ACM 40th International Conference on Software Engineering: New Ideas and Emerging Technologies Results*. 17–20.

[63] Miqing Li, Tao Chen, and Xin Yao. 2020, in press. How to Evaluate Solutions in Pareto-based Search-Based Software Engineering? A Critical Review and Methodological Guidance. *IEEE Transactions on Software Engineering* (2020, in press). https://doi.org/10.1109/TSE.2020.3036108

[64] Miqing Li, Crina Grosan, Shengxiang Yang, Xiaohui Liu, and Xin Yao. 2018. Multiline Distance Minimization: A Visualized Many-Objective Test Problem Suite. *IEEE Trans. Evol. Comput.* 22, 1 (2018), 61–78. https://doi.org/10.1109/TEVC.2017.2655451

[65] Miqing Li, Shengxiang Yang, and Xiaohui Liu. 2014. Shift-Based Density Estimation for Pareto-Based Algorithms in Many-Objective Optimization. *IEEE Trans. Evol. Comput.* 18, 3 (2014), 348–365. https://doi.org/10.1109/TEVC.2013.2262178

[66] Miqing Li, Shengxiang Yang, Xiaohui Liu, and Ruimin Shen. 2013. A Comparative Study on Evolutionary Algorithms for Many-Objective Optimization. In *Proceedings of the 7th International Conference on Evolutionary Multi-Criterion Optimization (EMO)*. 261–275.

[67] Miqing Li and Xin Yao. 2019. Quality Evaluation of Solution Sets in Multiobjective Optimisation: A Survey. *Comput. Surveys* 52, 2 (2019).

[68] Miqing Li and Xin Yao. 2020. What weights work for you? adapting weights for any pareto front shape in decomposition-based evolutionary multiobjective optimisation. *Evolutionary Computation* 28, 2 (2020), 227–253.

[69] Xiaoli Lian, Li Zhang, Jing Jiang, and William Goss. 2018. An approach for optimized feature selection in large-scale software product lines. *Journal of Systems and Software* 137 (2018), 636–651.

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

[71] Usman Mansoor, Marouane Kessentini, Manuel Wimmer, and Kalyanmoy Deb. 2015. Multi-view refactoring of class and activity diagrams using a multi-objective evolutionary algorithm. *Software Quality Journal* 25, 2 (2015), 1–29.

[72] Thainá Mariani and Silvia Regina Vergilio. 2017. A systematic review on search-based refactoring. *Inf. Softw. Technol.* 83 (2017), 14–34. https://doi.org/10.1016/j.infsof.2016.11.009

[73] Goran Mausa, Tihana Galinac Grbac, Bojana Dalbelo Basic, and Mario-Osvin Pavcevic. 2013. Hill Climbing and simulated annealing in large scale next release problem. In *Proceedings of Eurocon 2013, International Conference on*

Computer as a Tool, Zagreb, Croatia, July 1-4, 2013. IEEE, 452–459. https://doi.org/10.1109/EUROCON.2013.6625021

[74] Daniel A. Menascé, Daniel Barbará, and Ronald Dodge. 2001. Preserving QoS of e-commerce sites through self-tuning: a performance model approach. In *Proceedings 3rd ACM Conference on Electronic Commerce (EC-2001), Tampa, Florida, USA, October 14-17, 2001*, Michael P. Wellman and Yoav Shoham (Eds.). ACM, 224–234. https://doi.org/10.1145/501158.501186

[75] Nikolaos Mittas and Lefteris Angelis. 2013. Ranking and Clustering Software Cost Estimation Models through a Multiple Comparisons Algorithm. *IEEE Trans. Software Eng.* 39, 4 (2013), 537–551.

[76] Mohamed Wiem Mkaouer, Marouane Kessentini, Slim Bechikh, Mel Ó Cinnéide, and Kalyanmoy Deb. 2016. On the use of many quality attributes for software refactoring: a many-objective search-based software engineering approach. *Empirical Software Engineering* 21, 6 (2016), 2503–2545.

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

[78] Yang Nan, Ke Shang, Hisao Ishibuchi, et al. 2019. A Study of the Naïve Objective Space Normalization Method in MOEA/D. In *2019 IEEE Symposium Series on Computational Intelligence (SSCI)*. IEEE, 1834–1840.

[79] Rafael Olaechea, Derek Rayside, Jianmei Guo, and Krzysztof Czarnecki. 2014. Comparison of exact and approximate multi-objective optimization for software product lines. In *International Software Product Line Conference*. 92–101.

[80] Dipesh Pradhan, Shuai Wang, Shaukat Ali, and Tao Yue. 2016. Search-Based Cost-Effective Test Case Selection within a Time Budget: An Empirical Study. In *Proceedings of the 2016 on Genetic and Evolutionary Computation Conference, Denver, CO, USA, July 20 - 24, 2016*, Tobias Friedrich, Frank Neumann, and Andrew M. Sutton (Eds.). ACM, 1085–1092.

[81] K Praditwong, M Harman, and Xin Yao. 2011. Software Module Clustering as a Multi-Objective Search Problem. *Software Engineering IEEE Transactions on* 37, 2 (2011), 264–282.

[82] Raghu Ramakrishnan and Arvinder Kaur. 2020. Performance evaluation of web service response time probability distribution models for business process cycle time simulation. *J. Syst. Softw.* 161 (2020). https://doi.org/10.1016/j.jss.2019.110480

[83] Aurora Ramirez, José Raúl Romero, and Sebastian Ventura. 2019. A survey of many-objective optimisation in search-based software engineering. *Journal of Systems and Software* 149 (2019), 382–395.

[84] Andres J. Ramirez, Betty H. C. Cheng, Philip K. McKinley, and Benjamin E. Beckmann. 2010. Automatically generating adaptive logic to balance non-functional tradeoffs during reconfiguration. In *Proceedings of the 7th International Conference on Autonomic Computing, ICAC 2010, Washington, DC, USA, June 7-11, 2010*, Manish Parashar, Renato J. O. Figueiredo, and Emre Kiciman (Eds.). ACM, 225–234.

[85] Andres J. Ramirez, David B. Knoester, Betty H. C. Cheng, and Philip K. McKinley. 2009. 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*. 97–106.

[86] Abdel Salam Sayyad and Hany Ammar. 2013. Pareto-optimal search-based software engineering (POSBSE): A literature survey. In *The 2nd International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering (RAISE)*. IEEE, 21–27.

[87] Abdel Salam Sayyad, Joseph Ingram, Tim Menzies, and Hany Ammar. 2013. Optimum feature selection in software product lines: Let your model and values guide your search. In *International Workshop on Combining Modelling and Search-Based Software Engineering*. 22–27.

[88] Abdel Salam Sayyad, Joseph Ingram, Tim Menzies, and Hany Ammar. 2013. Scalable product line configuration: A straw to break the camel's back. In *IEEE/ACM International Conference on Automated Software Engineering*. 465–474.

[89] Abdel Salam Sayyad, Tim Menzies, and Hany Ammar. 2013. On the value of user preferences in search-based software engineering: A case study in software product lines. In *International Conference on Software Engineering*. 492–501.

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

[91] Dalia Sobhy, Leandro L. Minku, Rami Bahsoon, Tao Chen, and Rick Kazman. 2020. Run-time evaluation of architectures: A case study of diversification in IoT. *Journal of Systems and Software* 159 (2020). https://doi.org/10.1016/j.jss.2019.110428

[92] Matt Staats, Gregory Gay, Michael W. Whalen, and Mats Per Erik Heimdahl. 2012. On the Danger of Coverage Directed Test Case Generation. In *Fundamental Approaches to Software Engineering - 15th International Conference, FASE 2012, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2012, Tallinn, Estonia, March 24 - April 1, 2012. Proceedings (Lecture Notes in Computer Science, Vol. 7212)*, Juan de Lara and Andrea Zisman (Eds.). Springer, 409–424. https://doi.org/10.1007/978-3-642-28872-2_28

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

[94] Hiroshi Wada, Junichi Suzuki, Yuji Yamano, and Katsuya Oba. 2012. E3: A Multiobjective Optimization Framework for SLA-Aware Service Composition. *IEEE Transactions on Services Computing* 5, 3 (2012), 358–372.

[95] Florian Wagner, Adrian Klein, Benjamin Klopper, Fuyuki Ishikawa, and Shinichi Honiden. 2012. Multi-objective Service Composition with Time- and Input-Dependent QoS. In *IEEE International Conference on Web Services*. 234–241.

[96] Tobias Wagner, Nicola Beume, and Boris Naujoks. 2007. Pareto-, Aggregation-, and Indicator-Based Methods in Many-Objective Optimization. In *Proceedings of the 4th International Conference on Evolutionary Multi-Criterion Optimization (EMO)*. 742–756.

[97] Kewen Wang, Xuelian Lin, and Wenzhong Tang. 2012. Predator - An experience guided configuration optimizer for Hadoop MapReduce. In *4th IEEE International Conference on Cloud Computing Technology and Science Proceedings, CloudCom 2012, Taipei, Taiwan, December 3-6, 2012*. IEEE Computer Society, 419–426. https://doi.org/10.1109/CloudCom.2012.6427486

[98] Shuai Wang, Shaukat Ali, Tao Yue, and Marius Liaaen. 2018. Integrating Weight Assignment Strategies With NSGA-II for Supporting User Preference Multiobjective Optimization. *IEEE Trans. Evol. Comput.* 22, 3 (2018), 378–393.

[99] Zhenkun Wang, Yew-Soon Ong, Jianyong Sun, Abhishek Gupta, and Qingfu Zhang. 2018. A generator for multi-objective test problems with difficult-to-approximate Pareto front boundaries. *IEEE Transactions on Evolutionary Computation* 23, 4 (2018), 556–571.

[100] Frank Wilcoxon. 1945. Individual Comparisons by Ranking Methods.

[101] Tianpei Xia, Rahul Krishna, Jianfeng Chen, George Mathew, Xipeng Shen, and Tim Menzies. 2018. Hyperparameter Optimization for Effort Estimation. *CoRR* abs/1805.00336 (2018). arXiv:1805.00336 http://arxiv.org/abs/1805.00336

[102] Yinxing Xue and Yan-Fu Li. 2020. 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.* 29, 3 (2020).

[103] Shengxiang Yang, Miqing Li, Xiaohui Liu, and Jinhua Zheng. 2013. A grid-based evolutionary algorithm for many-objective optimization. *IEEE Transactions on Evolutionary Computation* 17, 5 (2013), 721–736.

[104] Qingfu Zhang and Hui Li. 2007. MOEA/D: A Multiobjective Evolutionary Algorithm Based on Decomposition. *IEEE Trans. Evol. Comput.* 11, 6 (2007), 712–731.

[105] Yuanyuan Zhang, Mark Harman, and S. Afshin Mansouri. 2007. The multi-objective next release problem. In *Proceedings of the Genetic and Evolutionary Computation Conference*. 1129–1137.

[106] Yuanyuan Zhang, Mark Harman, Gabriela Ochoa, Guenther Ruhe, and Sjaak Brinkkemper. 2018. An Empirical Study of Meta- and Hyper-Heuristic Search for Multi-Objective Release Planning. *ACM Trans. Softw. Eng. Methodol.* 27, 1 (2018), 3:1–3:32.

[107] Zibin Zheng, Yilei Zhang, and Michael R Lyu. 2012. Investigating QoS of real-world web services. *IEEE transactions on services computing* 7, 1 (2012), 32–39.

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

[109] Eckart Zitzler and Lothar Thiele. 1998. Multiobjective Optimization Using Evolutionary Algorithms - A Comparative Case Study. In *Parallel Problem Solving from Nature - PPSN V, 5th International Conference, Amsterdam, The Netherlands, September 27-30, 1998, Proceedings (Lecture Notes in Computer Science, Vol. 1498)*, A. E. Eiben, Thomas Bäck, Marc Schoenauer, and Hans-Paul Schwefel (Eds.). Springer, 292–304. https://doi.org/10.1007/BFb0056872