FEMOSAA: Feature-Guided and Knee-Driven Multi-Objective Optimization for Self-Adaptive Software

TAO CHEN, Department of Computing and Technology, Nottingham Trent University, UK, and CERCIA, School of Computer Science, University of Birmingham, UK
KE LI, School of Computer Science and Engineering, University of Electronic Science and Technology of China, China, and Department of Computer Science, University of Exeter, UK
RAMI BAHSOON, CERCIA, School of Computer Science, University of Birmingham, UK
XIN YAO, Department of Computer Science and Engineering, Southern University of Science and Technology, Shenzhen, China, and CERCIA, School of Computer Science, University of Birmingham, UK

Self-Adaptive Software (SAS) can reconfigure itself to adapt to the changing environment at runtime, aiming to continually optimize conflicted nonfunctional objectives (e.g., response time, energy consumption, throughput, cost, etc.). In this article, we present Feature-guided and knEe-driven Multi-Objective optimization for Self-Adaptive softwAre (FEMOSAA), a novel framework that automatically synergizes the feature model and Multi-Objective Evolutionary Algorithm (MOEA) to optimize SAS at runtime. FEMOSAA operates in two phases: at design time, FEMOSAA automatically transposes the engineers' design of SAS, expressed as a feature model, to fit the MOEA, creating new chromosome representation and reproduction operators. At runtime, FEMOSAA utilizes the feature model as domain knowledge to guide the search and further extend the MOEA, providing a larger chance for finding better solutions. In addition, we have designed a new method to search for the knee solutions, which can achieve a balanced tradeoff. We comprehensively evaluated FEMOSAA on two running SAS: One is a highly complex SAS with various adaptable real-world software under the realistic workload trace; another is a service-oriented SAS that can be dynamically composed from services. In particular, we compared the effectiveness and overhead of FEMOSAA against four of its variants and three other search-based frameworks for SAS under various scenarios, including three commonly applied MOEAs, two workload patterns, and diverse conflicting quality objectives. The results reveal the effectiveness of FEMOSAA and its superiority over the others with high statistical significance and nontrivial effect sizes.

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

Additional Key Words and Phrases: Feature model, search-based software engineering, multi-objective evolutionary algorithm, multi-objective optimization, self-adaptive system, performance engineering

This work is supported by the Ministry of Science and Technology of China (Grant No. 2017YFC0804003), Science and Technology Innovation Committee Foundation of Shenzhen (Grant No. ZDSYS201703031748284), and EPSRC (Grant Nos. EP/J017515/01 and EP/K001523).

Authors' addresses: T. Chen (co-corresponding author), Nottingham Trent University, Nottingham, UK, NG11 8NS and University of Birmingham, Birmingham, UK. B15 2TT; email: t.chen@cs.bham.ac.uk; K. Li (co-corresponding author), University of Electronic Science and Technology of 5 China, Chengdu, China and University of Exeter, Exeter, UK, EX4 4QD; email: keli.genius@gmail.com; R. Bahsoon, University of Birmingham, Birmingham, UK, B15 2TT; email: r.bahsoon@cs.bham.ac.uk; X. Yao (co-corresponding author), Southern University of Science and Technology, Shenzhen, China, 518055 and University of Birmingham, Birmingham, UK, B15 2TT; email: xiny@sustc.edu.cn.

ACM 1049-331X/2018/06-ART5
https://doi.org/10.1145/3204459

ACM Reference format:
Tao Chen, Ke Li, Rami Bahsoon, and Xin Yao. 2018. FEMOSAA: Feature-Guided and Knee-Driven Multi-Objective Optimization for Self-Adaptive Software. *ACM Trans. Softw. Eng. Methodol.* 27, 2, Article 5 (June 2018), 50 pages.
https://doi.org/10.1145/3204459

1 INTRODUCTION

Self-Adaptive Software (SAS) is a special type of software that is capable of adapting and reconfiguring itself at runtime through a set of known features (e.g., CPU cap, thread pool size, cache size, etc.), according to the changing environment [17]. One major goal of SAS is to continually optimize multiple and often conflicting nonfunctional objectives (e.g., response time versus energy consumption, throughput versus cost, and the like). However, given the dynamic and uncertain nature of running software, it is difficult to fully specify all possible conditions and their adaptation solutions at design time. Thus, designing an efficient and effective runtime optimization approach is necessary yet challenging. Depending on the complexity of SAS, software engineers have exploited various search algorithms (e.g., exact or stochastic search) for continually finding the optimal (or near-optimal) adaptation solution for SAS at runtime [23][51][12][42][41][15][13].

To optimize SAS at runtime using the search algorithms, there are two crucial challenges: First, it is difficult to effectively and systematically convert the SAS design to the context of a search algorithm while considering the right encoding of features in the representation of optimization (e.g., using only the features that contribute to different aspects of the variability of SAS). Here, the features might be categorical or numeric, where the former refers to those with distinct characteristics (e.g., the *Cache* feature is "on" or "off"), and the latter denotes those that can be quantified, measured, and sorted (e.g., the size of *maxThreads*). Furthermore, it is difficult to effectively and systematically handle the features' dependencies; for example, one can change *Cache Mode* only if the *Cache* feature is "turned on." Dependency can become even more complex in the presence of numeric features; for example, in Tomcat [2], the size of *maxThreads* should not be less than the size of *minSpareThreads*. Those conversion tasks are nontrivial as the design of SAS can be complex, and most search algorithms cannot handle dependency constraints in nature. Second, optimizing multiple conflicting objectives and managing their tradeoffs are complex and challenging, especially for SAS runtime. This is attributed to the huge number of alternative adaptation solutions and the requirement that the found solution be effective. Moreover, the dynamic and uncertain nature of SAS further complicates the conflicting relations between objectives, rendering the tradeoff surface difficult to explore. Those challenges, when not appropriately addressed, can result in compromised quality, unacceptable running overhead, and imbalanced tradeoff in SAS runtime optimization.

Most existing work fails to handle the first challenge as researchers have relied on a manual and/or incomplete conversion of the SAS design in the search algorithm's context [42][1][22][51], which renders the process expensive, nonsystematic, and error-prone. Moreover, the feature dependencies are often ignored, wasting the valuable function evaluations on invalid solutions at SAS runtime while providing no guarantee of finding the valid ones. Inspired by the applications of search algorithms to Software Product Line problems [45], researchers [23][41] have combined the feature model [33] with search algorithms to optimize SAS at runtime, considering categorical dependencies. However, numeric features are ignored, and a solution often encodes all the features using a simple binary representation. This might lead to the *curse of dimensionality* and thereby entail unnecessary complexity at SAS runtime. Further, existing approaches cannot prevent wasteful exploration of invalid solutions and difficult-to-handle dependencies related to numeric features.

For the second challenge, exact search [23] [9], with the helps of objective aggregation (e.g., a weighted sum), has been exploited for SAS runtime optimization. However, modern SAS often exhibits high variability, leading to an explosion of the search space of all possible solutions and rendering the problem intractable. Henceforth, exact search may fail to scale at runtime. In contrast, stochastic search, particularly evolutionary algorithms that are widely applied in Search-Based Software Engineering (SBSE), tend to be naturally robust in solving problems with extremely high numbers of alternatives and thus appealing for SAS optimization [29]. Those algorithms, when properly tailored, can lead to approximate and near-optimal solutions for complex software engineering problems with reasonable running time (minutes, if not seconds) [31]. Furthermore, stochastic search has proved effective for many real-time systems [22][27][51][12]. Often, existing approaches rely on a single-objective evolutionary algorithm to optimize SAS by simply transforming a multi-objective problem into an aggregated single-objective one [42][27]. While objective aggregation might be preferable for some contexts, it has been shown that there are cases where assigning weights to different objectives is a nontrivial task for software engineers, and the aggregation can hardly maintain a good diversity of solutions [29]. To alleviate this issue, studies [1][22][51] have used NSGA-II [20], a popular Multi-Objective Evolutionary Algorithm (MOEA), to optimize SAS without using weighted aggregation; these have shown that MOEA can find more convergent and diverse solutions in the tradeoff surface than optimizing via objective aggregation. However, NSGA-II has a coarse diversity preservation mechanism that is unable to provide well-distributed solutions in certain cases [52]. Therefore, it is desirable to have a general framework that can easily work with different MOEAs for optimizing SAS without suffering the limitation of one specific algorithm. In addition, given the fact that MOEAs produce a set of nondominated solutions, there is no established method for the SAS to choose an appropriate one for adaptation at runtime, thus entailing the risk of imbalanced tradeoffs.

To address these challenges and limitations, this article presents Feature-guided and knEe-driven Multi-Objective optimization for Self-Adaptive softwAre (FEMOSAA), a novel framework that automatically synergizes the feature model and a given MOEA to optimize SAS at runtime. Specifically, our contributions include:

—We rely on the feature model to represent the design of a given SAS with explicit considerations of numeric features and their dependencies. In FEMOSAA, we provide an automatic and systematic approach to transpose a given design of SAS, expressed as a feature model, into the MOEA's context at design time. Further, such transposition extends the internal structure of MOEAs in order to improve their ability to search for better adaptation solutions at SAS runtime. Notably, we contribute to the following in the transposition approach:

(1) To tailor the problem to be more suitable for SAS runtime, we discard lengthy binary encoding. Instead, our approach identifies the ***elitist features*** from the feature model to encode an elegant and polyadic chromosome representation in the MOEA. By "elitist features," we refer to those that cannot be removed in the optimization without damaging the original variability of SAS while minimizing the length of encoding. The benefit of such encoding is that (i) it is intuitive, simpler, and enables direct dependency extraction and (ii) reducing the number of genes helps to greatly shrink the search space and simplify the dependency constraints, which also improves the quality of the solutions found while shortening the running time of MOEA.

(2) To better guide the search and avoid exploring invalid solutions, our approach extracts feature dependencies with respect to these elitist features. Then, these dependencies are injected into the basic mutation and crossover operators of the MOEA to create new dependency-aware operators. These operators can systematically steer the MOEA to focus on exploring valid solutions for SAS, creating a larger chance to find better ones.

— Without loss of generality, we design FEMOSAA in such a way that it can be seamlessly integrated with different MOEAs[1] to optimize SAS at runtime. The elitist features and extracted dependencies, as processed by the transposition approach at design time, are used to guide the running behaviors of a given MOEA for SAS runtime optimization. In this work, we run FEMOSAA with three fundamentally distinct yet widely used MOEAs in the literature: MOEA-based Decomposition with STable-Matching model (MOEA/D-STM) [36], Non-dominated Sort Genetic Algorithm-II (NSGA-II) [20], and Indicator-Based Evolutionary Algorithm (IBEA) [53].
— To achieve a balanced tradeoff in SAS optimization, FEMOSAA identifies knee solutions automatically from the final nondominated set. The knee solutions often imply well-balanced tradeoffs, such that any improvement on one objective of a knee will cause relatively severe degradations on others.
— We conduct comprehensive experiments on two running SAS: One is a highly complex SAS that consists of the *eBay*-like RUBiS benchmark [43] and a set of real-world adaptable software (i.e., Apache Tomcat [2], MySQL [40], Ehcache [3], and Xen [48]) under the realistic FIFA98 workload trace [5]; another is a service-oriented SAS that can be dynamically composed by various services. We compare FEMOSAA with four of its variants (e.g., without dependency-aware operators) and three other state-of-the-art frameworks (i.e., DUSE [1], PLATO [42], and FUSION [23]) under various scenarios, including three commonly applied MOEAs (i.e., MOEA/D-STM, NSGA-II, and IBEA) and two different workload patterns[2] (i.e., read-write and read-only) along with diverse conflicting quality objectives. The experiments reveal the effectiveness of FEMOSAA and its superiority over the others when optimizing conflicting objectives for SAS, with statistically significant results and nontrivial effect sizes.

The contributions have clear impact on the synergy between software engineering for SAS and evolutionary computation as FEMOSAA combines strengths from both fields. Unlike many SBSE work that simply formulates the software engineering problem as a classic optimization problem for some MOEAs, our deeper synergy takes one step further by automatically and dynamically extracting the domain information of SAS to extend the internal structure of MOEA, thus improving its search ability. As a result, to control and exploit the power of MOEAs, SAS software engineers only need to provide the feature model when using FEMOSAA, without being an expert on MOEA. In addition, FEMOSAA improves MOEA and provides insights for MOEA researchers to design better algorithms for SAS since the identified elitist features and their dependencies serve as the engineers' systematic domain knowledge by which we can reduce the search space and better guide the search, providing a larger chance for finding better solutions.

The remainder of this article is organized as follows: Section 2 illustrates a detailed motivating example of SAS. Section 3 presents the background and extended notions of numeric features in the feature model. Section 4 gives an overview of FEMOSAA. Section 5 illustrates our approach that transposes a feature model to MOEA. Section 6 presents how the internal structure of existing MOEAs can be extended to combine with our dependency-aware operators and knee selection. Experimental results, verifiability, and threats to validity are discussed in Section 7. Finally, Sections 8 and 9 present related work and a conclusion, respectively.

[1] In addition to MOEAs, FEMOSAA also works with single-objective evolutionary algorithms in which case the knee selection method would be deactivated.
[2] Different workload patterns will create diverse behaviors of the SAS.

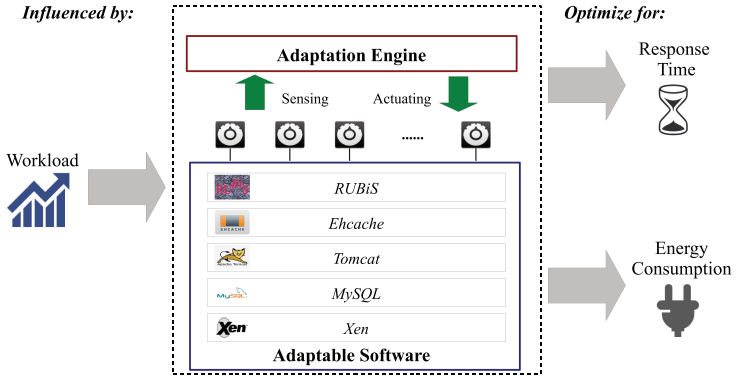


Fig. 1. An example of SAS.

2 A DETAILED MOTIVATING SCENARIO OF SELF-ADAPTIVE SOFTWARE

While our work can be applied to different contexts that demand runtime adaptation, we draw on a representative and realistic SAS to motivate and illustrate the need. As shown in Figure 1, like many SASs, the SAS example consists of two parts: an adaptable software that is being managed at runtime and an engine that controls the adaptation. Additionally, the SAS contains a complex software stack consisting of RUBiS[3] [43], Apache Tomcat [2], Ehcache [3], and MySQL [40], running on the virtualization hypervisor Xen [48]. The RUBiS benchmark serves as a representative of many real-world software applications that offer diverse functionalities and services to many end-users concurrently. We can see from Figure 1, as is the case of most practical software applications, the SAS's software stack contains a large amount off-the-shelf real-world software. Each of the software items support various control features, which, together with those from other software in the stack, can be changed dynamically on-the-fly to influence the runtime behaviors of the software system. An example of the control features includes the number of threads, the memory allocation, and enabling/disabling cache mechanism. By design, all possible configurations of control features form the search space or variability of the SAS. As the workload changes, the SAS is capable of adapting features at runtime to optimize for various nonfunctional quality attributes (e.g., response time). To achieve such goal, and thanks to the rapid development of search algorithms, SAS is often designed to continually search for that combination of feature configurations that leads to optimal (or near optimal) quality at runtime. However, to effectively and efficiently engineer SAS in this way is challenging for the following reasons:

Encoding the Features from the SAS Design. Consider a complex SAS which contains many features and configurations, systematically and generically choosing the right features and encoding them into the representation of search algorithm is difficult. To optimize the SAS at runtime, such representation defines the fundamental search space of the problem to be explored; therefore, the encoding of features could have a positive or negative impact on the search ability of a potential search algorithm. Given that some features in the SAS design do not contribute to the SAS's variability or can represent the same aspect of variability [6], existing work [23] [41] that simply encodes all features in a binary format is unnecessary. Consider a feature model with 100 features, binary representation can easily create a search space of 2^{100} and this, as we will show in Section 7.4.1, can negatively affect adaptation quality and overhead.

Handling Dependencies in the SAS Design. Many widely used exact and stochastic search algorithms (e.g., MOEA) are not designed to handle dependency constraints. This makes the

[3]An eBay-like software application with 26 services.

treatment of dependencies difficult, especially when the dependencies in SAS come in a mixture of categorical dependencies (e.g., *Cache Mode require Cache*) and numeric ones (e.g., *maxThreads* ≥ *minSpareThreads*). As we will show in Section 7.4.2, those dependencies, when ignored [42] [1] [27] or incorrectly handled [23] [41] (as in existing work), can degrade the adaptation quality.

Explosion of the Search Space. Modern SAS often has high variability, leading to an explosion of the search space. For example, the original design of the SAS shown in Figure 1 has a search space of more than 1 billion, which we will elaborate in detail in Section 3.3.

Tradeoff on Conflicting Objectives. SAS often exhibits multiple conflicting quality objectives that need to be optimized simultaneously, and tradeoffs need to be made. In general, many existing approaches [42] have assumed that the relative importance of objectives can be correctly quantified as numeric weights, which has been found to be difficult in some cases [29]. Those weights, when inappropriately specified and expressed, inevitably create negative impact on the search process and result in unwanted, poor adaptation quality. It is even more difficult to achieve balanced tradeoff.

These difficulties motivate our work, which automatically synergizes the feature model of SAS and a given MOEA, creating a feature-guided MOEA with knee selection to optimize SAS at runtime.

3 BACKGROUND AND PRELIMINARIES

3.1 Multi-Objective Evolutionary Algorithm (MOEA)

Evolutionary algorithm, a stochastic search-based meta-heuristic, has been widely accepted as a major approach for solving multi-objective optimization problems [19], in which case it is also known as MOEA. In MOEA, the population contains a set of solutions (individuals), each of which is represented by a fixed-length thread-like chromosome carrying different values at each gene. As shown in Figure 2 and Algorithm 1, the evolutionary search of MOEA starts after the initialization of the population (Lines 2 to 9). During the search process, the elite information can propagate from parents to offspring via some random and probabilistic reproduction operations (i.e., crossover and mutation) on the mating parents chosen from the mating selection procedure. Inspired by the *survival of the fittest* rule from evolutionism, survival selection preserves high-quality individuals with superior fitness values to the next iteration (generation), as shown in Lines 10 to 24. The evolution process repeats until a stopping criteria (e.g., a predefined function evaluation threshold) is satisfied. The major difference between MOEA and the classic single-objective evolutionary algorithm lies in the mating and survival selection mechanisms. In particular, instead of finding a single optimal (or near optimal) solution, as in the single-objective evolutionary algorithm, MOEA aims to find a set of nondominated solutions[4] that approximate the *Pareto front* with good convergence and uniform distribution (Line 25). Notably, for every solution in the nondominated set, any improvement of an objective will result in a degradation for at least one other objective.

Generally, the existing MOEAs can be divided into the following three categories according to the survival selection mechanisms:

- *Decomposition-based method*: The MOEA decomposes the original multi-objective optimization problem into several single-objective optimization subproblems by linear or nonlinear aggregation methods [38]. Then, it uses a population-based technique to solve these subproblems in a collaborative manner. MOEA/D [52], MOEA/D-STM [36], and NSGA-III [18] are the representative algorithms of this sort.

[4]A solution dominates another if it has at least one objective better than another while all other objectives are not worse than another. Nondominated solutions denote those solutions that are not dominated by any other solutions in the set.

ALGORITHM 1: General Algorithmic Process of MOEA

Input: Given mutation rate r_m, crossover rate r_c and the maximum number of evaluation $eval_{max}$, which is often equivalent to the size of population × the maximum number of generations
Output: A set of optimized non-dominated solutions

```
start evolution
P = ∅
eval = 0
for i = 1 to P_size do
    S = getRandomSolution()
    evaluateFitness(S)
    eval = eval + 1
    P = P + S
end for
while eval < eval_max do
    P_0 := ∅
    while |P_0| ≤ P_size do
        parents := doMatingSelection(P)
        offspring := doCrossover(parents, r_c)
        for each solution S in offspring do
            doMutation(S, r_m)
        end for
        evaluateFitness(offspring)
        eval := eval + |offspring|
        P_0 := P_0 ∪ offspring
    end while
    P := P ∪ P_0
    doSurvivalSelection(P, P_size)
end while
return getNonDominatedSolutions(P)
end evolution
```

- *Pareto-based method*: The MOEA uses a Pareto dominance relation as the primary selection criterion to push solutions as close to the Pareto front as possible. Meanwhile, it employs some density estimation techniques (e.g., crowding distance [20] and clustering analysis [54]) to maintain population diversity. The representative algorithms are NSGA-II [20], SPEA2 [54], and PAES [34], and others.
- *Indicator-based method*: Here, sophisticated performance indicators are designed to measure the overall quality of a solution set. The representative algorithm is IBEA [53], which transfers the multi-objective optimization problem into a new single-objective one that aims to find the optimal set of solutions with respect to a given indicator.

3.2 Knee Solutions

The MOEA generates a set of nondominated solutions that approximate the Pareto front. However, not every nondominated solutions can lead to a balanced tradeoff for SAS runtime optimization. Indeed, the most common purpose of MOEA is to search and visualize a set of nondominated solutions that are as close to the true Pareto front as possible. Then, a human decision-maker can pick whichever solution that he or she prefers. However, there is no such human available in the SAS optimization problem. Therefore, a method is required to pick a sole solution from the resulting set of nondominated solutions to execute adaptation.

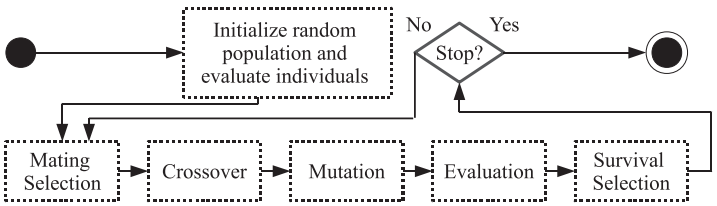


Fig. 2. The general workflow of MOEA.

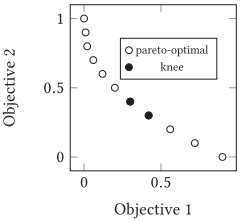


Fig. 3. Pareto optimal and knee solutions.

A simple Pareto optimal front is shown in Figure 3, where the two objectives should be minimized. Clearly, solutions near the edges strongly favor one objective over the other, but there is a visible bulge around the middle, which is the knee region. Those solutions in the knee region (or simply knee solutions) are characterized by the fact that a small improvement in either objective will cause a large deterioration in the other. In the case where human intervention is limited while the two objectives are equally important or when it is difficult to correctly weight them (which is common for SAS), knee solutions are more balanced than the others, and they are almost the most preferable ones. This is because the knee solutions achieve a good sense of compromise, while moving the solution in any direction from the knee region would create a bias toward an objective and lead to imbalanced adaptation results. Finding the knee solutions is challenging because real-world runtime SAS problems may not pose the perfect convex objective surface as shown in Figure 3.

3.3 Feature Model with Numeric Features

The feature model [16], expressed as the tree structure, is a widely used notation for software engineers to represent the functional variability of a software [6]. In feature-oriented domain analysis, the feature model is particularly important for expressing the possible variations under which a software system can operate in order to improve functional and nonfunctionary quality [33]. In this perspective, features define the prominent or distinctive aspects between different variations of a software system [33], which range from high-level architectural elements (an entire component) to low-level configurations (a specific parameter).

In the context of SAS, the inherited concept of a feature model allows it to define the extent to which the SAS is able to adapt at runtime (i.e., a range of variations that the SAS can achieve). Given this, some successful attempts have been made to apply the feature model to design SAS [23][41]. Therefore, to correctly exploit the feature model for SAS, the software engineer must identify (i) the variations of different features that are supported by the SAS and (ii) the dependency constraints that determine the validity of a given variation (adaptation solution). However, while the feature model is useful to express the variability of SAS (i.e., the search space of the adaptation decision-making problem), it does not correlate the effects of those variations to the concerned quality attributes. Therefore, in this work, we exploit an additional system model to evaluate how a variation can affect the quality of SAS, as we will discuss in Section 7.2.

Figure 4 shows an example of a feature model for one of the SASs we study in this article.[5] As we can see, there are four types of in-branch relation between a feature and its parent:

- *Optional* refers to the feature that might be deselected (e.g., *Cache*).
- *Mandatory* denotes core features that cannot be deselected (e.g., *Thread Pool*).

[5]In this article, we use a graphical figure of the feature model for more intuitive presentation. In practice, the feature model might be expressed in XML or conjunctive normal form, which can be parsed and analyzed directly by FEMOSAA.

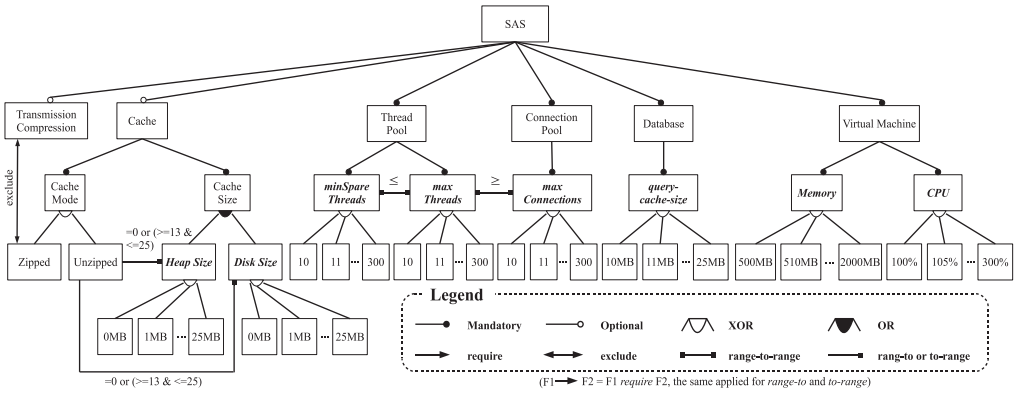


Fig. 4. The feature model for the example SAS shown in Figure 1. (The numeric features are shown in bold and italic letters. A zipped cache mode means the cached data are compressed, thereby costing a smaller amount of memory; otherwise, it is unzipped. CPU % denotes to what extent the SAS can consume CPU capacity, where each 100% means a CPU core, e.g., 200% means two cores; 150% means one full core and 50% of another).

- $\mathcal{XOR}$ represents the feature in a group such that exactly one group member can be selected (e.g., *Cache Mode*).
- $\mathcal{OR}$ means a group in which at least one group member needs to be selected (e.g., *Cache Size*).

When a feature is selected, it means that it is "turned on"; similarly, deselection of a feature means that it is "turned off." Selecting a feature implies that its parent should be selected, too. In this work, we call a feature ***deselectable*** if it has an *Optional*, $\mathcal{OR}$, or $\mathcal{XOR}$ relation to its parent or ***conditionally deselectable*** if it has a *Mandatory* relation to its parent but there exist deselectable ancestors. On the other hand, common cross-branch relations include:

- F_i *require* F_j means the former can only be selected if the latter is selected.
- F_i *exclude* F_j denotes two features that are symmetrically mutually exclusive.
- F_i *at-least-one-exist* F_j is an implied relation between the members of an $\mathcal{OR}$ group. It represents the same notion as that of $\mathcal{OR}$.
- F_i *at-least-one-require* F_j is an implied relation between a member of an $\mathcal{OR}$ group and another external feature, which has *require* at the root of the said $\mathcal{OR}$ group. It means F_i can only be selected if at least one member of the $\mathcal{OR}$ group, to which F_j belongs, is selected.

All these relations constitute the dependency chain(s) in the model. As in Figure 4, the number of features in the preceding example is 1,151, with a search space of more than a billion.

To better incorporate the feature model with SAS and simplify the design, we distinguish ***categorical features*** and ***numeric features***. We define numeric features as: *A feature is numeric if it has more than one child in its $\mathcal{XOR}$ group, and all its children can be quantified by real numbers.* For example, in Figure 4, *Memory* is clearly a numeric feature. Otherwise, the feature is categorical (e.g., *Cache Mode*). Similarly, a dependency is numeric as long as it is linked to numeric features and it involves quantitative comparisons. As in Figure 4, we propose the following cross-branch numeric dependencies for engineers to specify in their design:

- *Range-to-range*. This is associated with two numeric features, and it can be expressed as, for example, F_i *range-to-range* F_j ($F_i < F_j$), meaning that F_i's selected child in its $\mathcal{XOR}$ needs to

be smaller than that of F_j. It can be easily translated into a categorical dependency: $F_i < F_j$ simply means that F_i's $\mathcal{XOR}$ child C would have an *exclude* dependency on each F_j $\mathcal{XOR}$ child that is larger than or equal to the value of C. Other quantitative comparisons (e.g., >) can be also applied.

- *To-range*. This constrains a categorical feature F_i (dependent) with respect to a numeric feature F_j (main); for example, F_i *to-range* F_j ($F_j < 10$), meaning that F_i can only be selected if F_j's selected child in its $\mathcal{XOR}$ falls in the given range, as expressed by the mathematical formula. This can be translated to categorical dependency such that F_i would have *exclude* dependency on each of F_j's $\mathcal{XOR}$ children that is not in the range.
- *Range-to*. This is the inverse of *to-range* dependency where a numeric feature (dependent) is constrained by a categorical feature (main).

Clearly, numeric dependencies can only be cross-branched, while categorical ones exist on both in-branch and cross-branch. When a dependency is associated with one categorical feature and one numeric feature (i.e., *to-range* and *range-to*), we call it a *hybrid dependency,* which is a special case of numeric dependency. Note that numeric features might have all types of dependencies, but categorical features cannot be linked to the *range-to-range* numeric dependency.

3.3.1 The Benefits of Explicitly Considering Numeric Features. As mentioned, given that the feature model is discrete and statically defined at design time, it is possible to convert those numeric features and their dependencies into categorical ones without affecting the original variability of SAS. However, explicitly considering numeric features in the feature model will introduce the following benefits in terms of both design time analysis and runtime optimization in FEMOSAA:

- Explicitly considering the numeric features provides simpler and more intuitive design of the feature model as numeric features can be interpreted directly by the software engineers.
- Converting the numeric features into categorical ones will unnecessarily complicate the feature model, which can implicitly induce software engineers to design the feature model in a way that the children of numeric features would need to be encoded as genes. As mentioned, this will greatly increase the number of solution variables in the optimization, leading to the curse of dimensionality. Therefore, explicitly considering numeric features can provide us with the foundation to design novel and simpler encoding of chromosome representation in MOEA, as we will show in Section 5.1.
- Explicitly considering the numeric features results in fewer dependencies, in contrast to the case where the numeric features are converted into categorical ones. As we will show in Section 5.2, this simplifies our dependency extraction process for injecting dependencies into the mutation and crossover operators of MOEA. In addition, fewer dependencies implies simpler dependency structure; that is, a dependent feature has fewer main features, which in turn reduces the running overhead of our dependency-aware operators at runtime.

4 FEMOSAA OVERVIEW

As shown in Figure 5, a SAS generally consists of two parts: an adaptable software that is managed at runtime and an engine that controls the adaptation. The adaptable software could be a software stack that contains different interconnected software or middleware.

Our FEMOSAA framework is deployed as the adaptation engine, and it operates on both the design time and runtime of the SAS. At design time, FEMOSAA analyzes and transposes the feature model of SAS, which is provided by the software engineers, to the context of MOEA. The transposition first identifies elitist features (see Section 5.1), which are passed to the process for extracting the dependency to accommodate selected features (step 1), as we will explain in

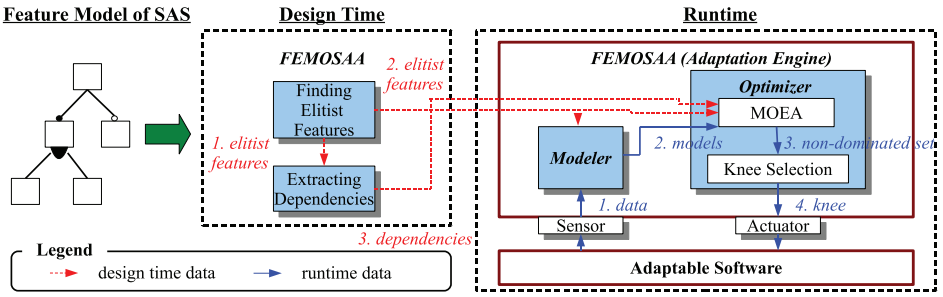


Fig. 5. The architecture of FEMOSAA.

Section 5.2. With the help of FEMOSAA, those elitist features and dependencies are stored and will be used directly by the MOEA at runtime (steps 2 and 3). Given that only the elitist features would be encoded into the chromosome representation of MOEA, the identified elitist features can be used as the objective functions' inputs and can serve as the indication of which sensors/actuators to use or to implement (step 2).

FEMOSAA has two main components at runtime: (i) a *Modeler* which contains the objectives (fitness functions) that build the correlation between features and quality attributes. Those objectives functions can be created using analytical models [44], simulation [26], or machine learning [10] [11] [14] in which they might be updated on-the-fly using data from sensors. And (ii) an *Optimizer* that realizes the MOEA (extended by our knee selection) and is guided by the transposed information from the feature model to find a single optimized solution for adaptation via actuators (see Section 6).

Given the uncertain and dynamic environment, these two components constitute the feedback loop that continually adapts the SAS toward better quality (e.g., improved response time). The adaptation cycle starts with monitoring the status of the SAS and the environment (step 1), which is then used to update the objective functions and model (step 2). Next, the feature-guided MOEA optimizes and searches for a set of nondominated solutions based on the updated objective functions (step 3), after which the knee selection selects the most balanced one for adaptation (step 4). The optimization can be triggered either by violations of quality requirements or, as in this work, by a fixed frequency (e.g., at a particular point in time). Note that we consider the execution order of a solution as a separate issue from the optimization. Thus, given a valid and optimized solution, we assume that the valid order of execution, with respect to the dependency, is enforced in the actuators through analyzing the dependencies in the feature model.

5 TRANSPOSING A FEATURE MODEL OF SELF-ADAPTIVE SOFTWARE TO MOEA AT DESIGN TIME

In this section, we present an automatic and systematic approach as part of FEMOSAA that transposes a feature model into MOEA's context. At design time, the approach finds the elitist features from the model (by which we refer to those that cannot be removed in the optimization without damaging the original variability of SAS while minimizing the length of encoding to form chromosome representation); it then extracts the feature dependency with respect to these elitist features. Such information will be used at runtime to guide the evolutionary optimization.

To guarantee correctness of the transposition, it is imperative to ensure that the feature model has been fully tested and verified by existing tools [6]. This ensures that faults (e.g., dead features, false options, and contradictory relations) have been already dealt with before the transposition. The verification of a feature model is beyond the scope of this work, however. Unlike our work,

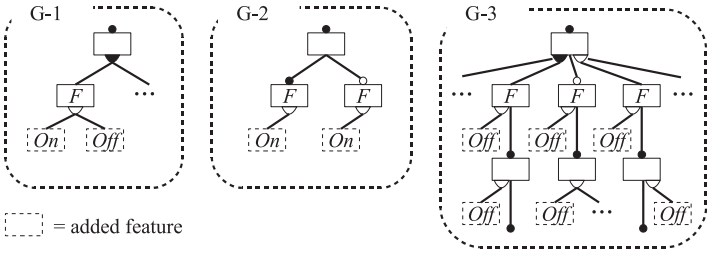


Fig. 6. The growing process in SAS's feature model.

the dependency related to numeric features is not treated explicitly in existing testing tools. However, as discussed in Section 3.3, a numeric (and hybrid) dependency can be easily transferred into a categorical dependency, which can be then tested directly. We also assume that all possible children (including *0*) of numeric features are discretized and predefined. It is worth noting that discretizing the numeric features is the first step to removing unnecessary complexity in our SAS optimization problem because many real-world features are often discrete and/or can be customized based on software engineers' knowledge (e.g., it may be known that changing memory allocation by less than 1MB does not affect the behavior and quality of a SAS; thus, instead of considering the memory feature as a continuous feature, the possible child features of the memory feature can be discretized at every 1MB).

While FEMOSAA is generic and can be applied to any case as long as the feature model and MOEAs are involved, in the following, we specify the transposition approach in FEMOSAA for general cases but refer to a concrete example for more intuitive illustration where appropriate. Specifically, in Section 5.1, we introduce an approach to identify the elitist chromosome representation of a SAS's feature model. Subsequently, in Section 5.2, we illustrate how the related dependency chains and the value trees can be extracted (Section 5.2.1) and merged (Section 5.2.2), according to the genes identified in Section 5.1.

5.1 Finding Elitist Features for Chromosome Representation

5.1.1 Growing the Feature Model Tree. Deselectable features in a feature model often do not explicitly indicate the "on" and "off" features as children, but they are important information for us to parse and understand the full variability of the model. Hence, to correctly transpose the feature model, we first grow the feature model tree to disclose the hidden information inferred from the deselectable features. As illustrated in Figure 6, this is achieved by adding children representing *On* and/or *Off* to any given feature F in the feature model using the following steps in order:

- **G-1.** If F is a leaf feature that has an $\mathcal{OR}$ relation to its parent, we then add two children representing *On* and *Off* in a $\mathcal{XOR}$ group to F. This explicitly states that, in such a case, the leaf F can have two mutually exclusive options, which is important to our encoding.
- **G-2.** If F is a leaf feature that has neither an $\mathcal{OR}$ nor $\mathcal{XOR}$ relation to its parent, we then add one child representing *On* in a $\mathcal{XOR}$ group to F. This ensures that every feature has the option of "on" (and translates them into branches to be parsed by **G-3**), except those with an $\mathcal{OR}$ or $\mathcal{XOR}$ relation to their parent, as the former has been considered in **G-1** while the latter's "on" option can be expressed by the parent.
- **G-3.** If F is a branch feature that has an *Optional*, $\mathcal{OR}$, or $\mathcal{XOR}$ relation to its parent, we then add one child representing *Off* in a $\mathcal{XOR}$ group to F and to the descendants of F that are

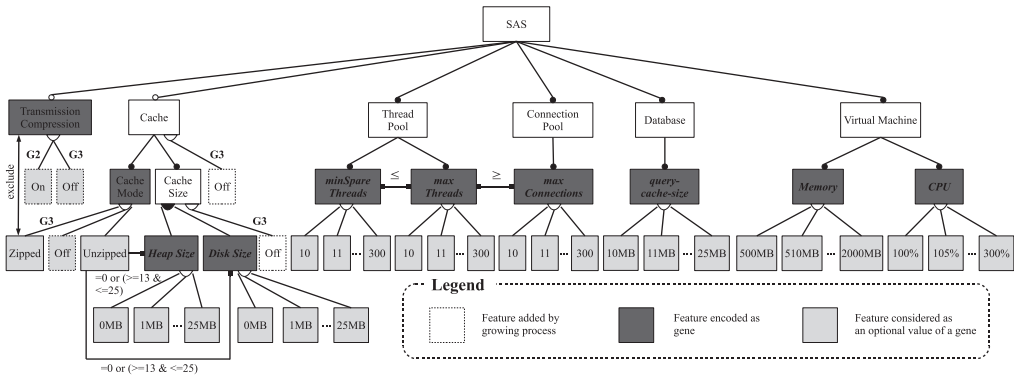


Fig. 7. The example SAS's feature model after the process of finding elitist features.

branch features (if they do not currently have a child representing *Off*). This ensures that both the deselectable and conditionally deselectable features expose the option of "off."

After growing the tree, the added features and the steps that create them are shown in Figure 7.

5.1.2 Identifying Genes from the Feature Model Tree. We have now obtained a model with no hidden information; the next phase is to find the elitist features for genotype encoding in MOEA, thus creating an ***elitist chromosome representation***. Intuitively, following the grown tree, our approach *encodes a feature F as a gene in the chromosome if and only if it is the parent of a $\mathcal{XOR}$ group, which contains more than one group member.* Hence, *F*'s children within the $\mathcal{XOR}$ group constitute its set of alternative optional values to be chosen in MOEA, subject to the constraints in dependencies. Drawing on this, the representation can be simplified in three aspects without affecting the original variability:

(1) Eliminating features whose variability can be expressed by their parent (i.e., those with $\mathcal{XOR}$ relations to the parent); for example, the variability of *CPU*'s children can be represented by itself.
(2) Eliminating features whose variability can be expressed by their descendants; for example, the variability of the *Cache* feature can be represented by the combination of *Cache Mode* and *Cache Size* features; *Cache Mode* can be represented by *Heap Size* and *Disk Size*.
(3) Eliminating those features that have no implication on variability (e.g., the *Thread Pool* is always mandatory). This, however, does not mean that we simply remove all mandatory features (as in [31]); instead, our approach retains those mandatory features with a $\mathcal{XOR}$ group of children as they would often help us to considerably reduce the number of genes, as explained in (1) above.

From now on, those features, which are chosen to be encoded in the chromosome, are called ***genes***. It is easy to see that numeric features are always chosen as genes. As shown in Figures 7 and 8, there are 10 features being considered as genes in the example feature model of SAS.[6] To make the model informative, we prune those features that are the only members of their corresponding $\mathcal{XOR}$ group. For all genes, if they select *Off* or *0* as their value, then it means they are deselected; any other values mean that they are selected. Note that when a gene selects *0*, it implies that the numeric value is 0 and that the feature is "turned off," which will have no further effects on the SAS.

[6]The other features, which are not genes, can be fixed to *On* in the SAS.

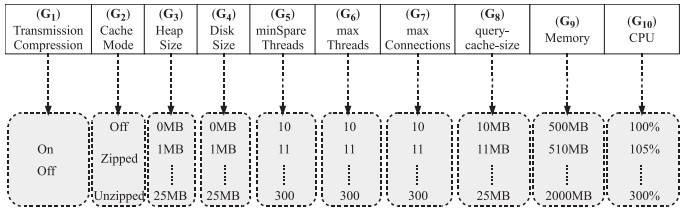


Fig. 8. The resulted chromosome representation of the SAS studied in form of genes ($G_1, G_2, \ldots, G_{10}$).

In this way, the elitist chromosome representation is polyadic, elegant, and free of unnecessary information (e.g., some unneeded relations to the parent of a feature), which is otherwise unavoidable in the classic binary representation. This, as we will show in Section 7.4.1, can bring nontrivial benefits for optimization quality and runtime overhead.

5.2 Extracting Feature Dependency for Guiding Evolution

5.2.1 Analyzing and Refactoring the Dependency. While the identified elitist chromosome representation can naturally prevent violation of the $\mathcal{XOR}$ relation, it does not contain any information about other dependency constraints (*require*, $\mathcal{OR}$, etc.). This issue is nontrivial as leaving it without treatment could result in a high possibility of exploring invalid solutions that negatively affect the quality of adaptation. To this end, our next step in the transposition is to extract and analyze the dependency chain(s) that accommodate the genes so that they can be injected into the mutation and crossover operators of MOEA to prevent the search from exploring an invalid solution. Here, a single dependency between two genes represents the constraint on the dependent gene with respect to the conditions of the main gene. The extracted dependencies and their imposed constraints are shown in the Table 1, which will be discussed in Section 5.2.2. Specifically, we distinguish two categories of dependency: *in-branch* and *cross-branch*.

Extracting in-branch dependency chain(s) aims to handle the constraints introduced by $\mathcal{Optional}$, $\mathcal{Mandatory}$, $\mathcal{OR}$, and $\mathcal{XOR}$ relations with respect to the genes. To achieve this, the features' in-branch dependencies are extracted in both vertical and horizontal directions while considering all the four relations.

Vertical analysis for extracting in-branch dependency helps to ensure that the in-branch relation between feature and parent is captured. As shown in Figure 9, for any feature F in the original feature model, we conduct the following vertical analysis:

- **VA-1.** If F is a gene and it is deselectable ($\mathcal{Optional}$, $\mathcal{OR}$, or $\mathcal{XOR}$ to its parent) or conditionally deselectable ($\mathcal{Mandatory}$ to its parent but has deselectable ancestors), then, for each path from F, the closest descendant gene D_g of F would have *require* dependency on F as D_g cannot be selected without the presence of F. Additionally, if D_g has $\mathcal{Mandatory}$ relation to its parent and the path between F and D_g does not contain deselectable features, then F would also have *require* on D_g as both features need to be selected at the same time.
- **VA-2.** In addition to **VA-1**, if F has $\mathcal{XOR}$ relation to its parent and it is a gene, then F would have a *require* on its parent, denoted as $P_g = \alpha$ (F's parent P_g would always be a gene, as ensured by our gene identification process), where α is the reference of F in P_g; similarly, $P_g = \alpha$ would also have *require* on F as both features need to be selected at the same time. On the other hand, if F has $\mathcal{XOR}$ relation to its parent but it is not a gene, then, for each path from F, the closest descendant gene D_g of F would have its own *require* on $P_g = \alpha$. Under the same case, if D_g has $\mathcal{Mandatory}$ relation to its parent and the path between F and D_g does not contain deselectable features, then $P_g = \alpha$ would also need to have *require* on D_g.

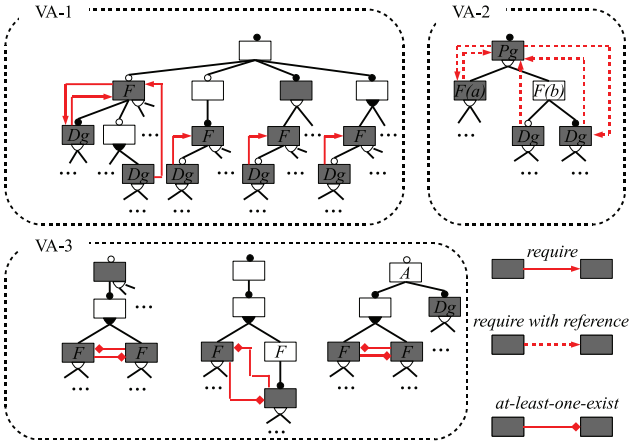


Fig. 9. The vertical analysis for extracting in-branch dependencies in SAS's feature model with respect to the elitist genes.

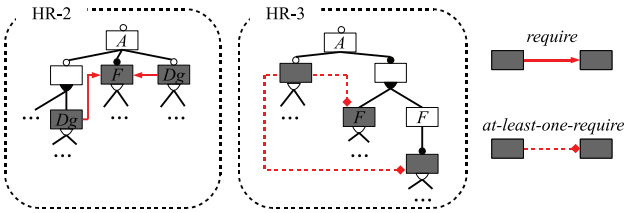


Fig. 10. The horizontal refactoring for extracting in-branch dependencies in SAS's feature model with respect to the elitist genes.

- **VA-3.** In addition to **VA-1**, if F has $\mathcal{OR}$ relation to its parent, we find the closest deselectable ancestor of F, denoted as A, if such an A does exist. Now, if at least one ancestor of F is a gene, or F's parent is neither deselectable nor conditionally deselectable, or there exists at least one closest descendant gene, D_g, of a path from A, such that D_g has *Mandatory* relation to its parent and there is no deselectable features in the path between D_g and A, then this means that the $\mathcal{OR}$ group for which F's parent is the root needs to select at least one group member. Thus, unless there already exist an *at-least-one-exist* dependency, F (if F is a gene) or its closest descendant genes, each of which follows different paths (if F is not a gene), would have *at-least-one-exist* on (i) the other closest descendant genes of F if it is not gene; (ii) those sibling genes of F in the same $\mathcal{OR}$ group; and (iii) the closest descendant genes, each of which follows different paths from F's siblings that are not genes but are in the same $\mathcal{OR}$ group as F.

The horizontal refactoring, on the other hand, is to ensure that elimination of some features does not mislead the dependencies implied by the original variability. Suppose that F is a feature in the original feature model and that A is the closest deselectable ancestor of F, if such an A does exist. Now, assuming that A is not a gene and that there is no gene on the path from A to F, we then conduct the following horizontal refactoring, as shown in Figure 10:

- **HR-1.** If F has *Optional* relation to its parent, then we do nothing, even if it is a gene. This is because the selection of F does not affect A's closest descendant genes, each of which follows the other paths from A.

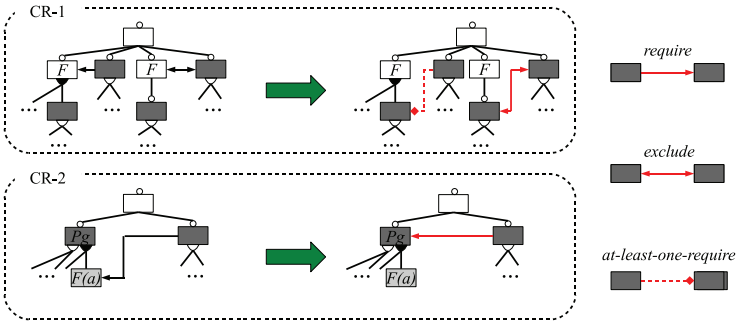


Fig. 11. The refactoring for extracting cross-branch dependencies in SAS's feature model with respect to the elitist genes.

- **HR-2.** If F is a gene that has *Mandatory* relation to its parent and it is conditionally deselectable, then, for each path from A, the closest descendant gene D_g of A (excluding F itself) would have *require* on F. This can ensure that, when D_g is selected, F would be also selected.
- **HR-3.** If F has $\mathcal{OR}$ relation to its parent and it does not have *at-least-one-exist* dependency for the group, then for each path from A (except the paths that pass through F's $\mathcal{OR}$ group), the closest descendant gene D_g of A would have *at-least-one-require* on F, if F is a gene; or on those closest descendant genes of F, each of which follows different paths from F, if F is not a gene. Hence, when D_g is selected, at least one member of the $\mathcal{OR}$ group of F (or their closest descendant genes) would be also selected.
- **HR-4.** If F has $\mathcal{XOR}$ relation to its parent, then we do nothing, even if it is a gene. This is because our gene identification process ensures that the parent of F would always be a gene, which also express the selection of F.

After considering the in-branch dependency, we now focus on refactoring the cross-branch dependency. If both sides of a cross-branch dependency are genes, then it can be extracted directly. However, if either side (or both) of the feature is not a gene, then a treatment is needed. Suppose that a feature F is associated with one or more cross-branch dependencies and that F is not a gene; we then do the following refactoring, as shown in Figure 11:

- **CR-1.** If F is a branch, then its cross-branch dependencies are migrated to those closest descendant genes of F, each of which follows different paths from F. Further, if F is the root of an $\mathcal{OR}$ group and it is the main feature in any *require* dependency, then those *requires* would be changed to *at-least-one-require*, which are migrated to the member genes of F's $\mathcal{OR}$ group, and to the closest descendant genes, each of which follows different paths from those members that are not genes.
- **CR-2.** If F is a leaf, then its cross-branch dependencies are migrated to the parent of F, denoted as P_g, where the dependency would remain the same but the main gene becomes $P_g = \alpha$, where α is the reference of F in P_g. Here, F would always have $\mathcal{XOR}$ relation to P_g because, if F was to have *Mandatory* relation to P_g, then there would be a contradiction as the main feature of a cross-branch dependency is mandatory. In addition, when F is a leaf, our growing process has ensured that F has neither $\mathcal{OR}$ nor *Optional* relation to P_g, which would always be a gene.

Finally, putting everything together, the extraction that occurs on the model and the extracted dependency chain(s), with respect to the elitist chromosome, are shown in Figures 12 and 13,

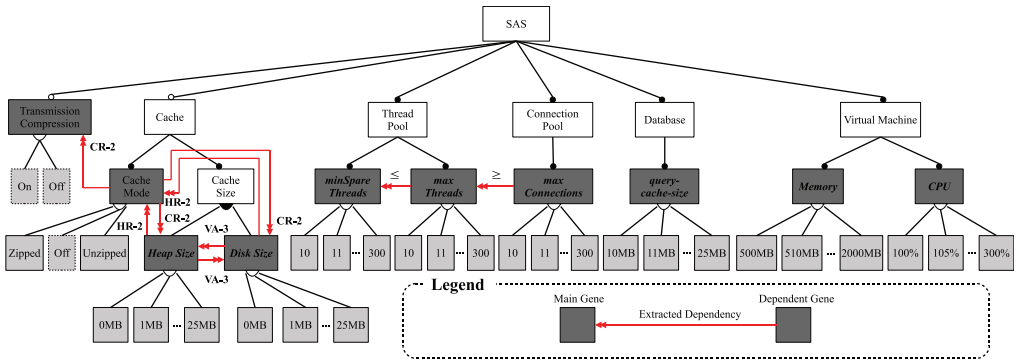


Fig. 12. The example SAS's feature model and the extracted dependency after the processes of VA, HR, and CR.

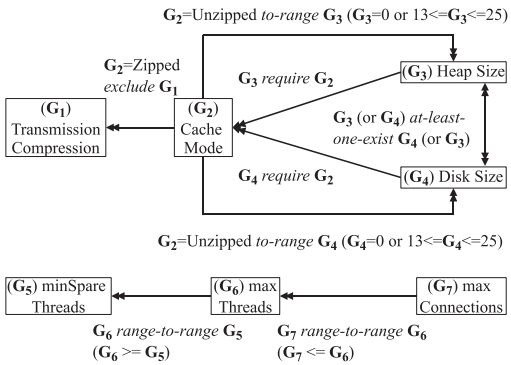


Fig. 13. The extracted dependency chains of genes for the example feature model.

respectively. The constraint of a dependent gene imposed by a dependency, according to Table 1, can be expressed using a ***value tree***, where each leaf is a set of optional values constrained by the corresponding condition in a branch (i.e., the selected values of the main gene). For example, in Figure 14, the value tree for the dependency between the *Cache Mode* (G_2) gene and *Transmission Compression* (G_1) gene constrains that the former can only be *Off* or *Unzipped* if the latter is *On*; or any optional values, otherwise. Note that if a gene is not a dependent of any dependency, then it would have a value tree without any branches.

5.2.2 Merging the Dependency. After the extraction, we can see that a dependent gene might have multiple dependencies on the same or different main genes. To construct a combined value tree for a dependent gene, the dependencies, by which it is constrained need to be merged one by one, using set operators (union or intersection) to combine the leaves from their value trees. Table 1 shows which set operators are needed for each dependency type when merging with the others; this is derived from the conjuncture normal form of the related genes.

Specifically, for every dependent gene, the merging process has the following steps:

—**Step 1.** If it has two or more dependencies with identical main genes, then the leaves, which are constrained by the same condition in the branches, would be combined directly using the set operations shown in Table 1.

Table 1. The Extracted Dependency Constraints Between Two Genes and the Related Set Operations for Merging Dependency

Dependency (denoted as D)	*Constraints on G_i*	*Merge with Other Dependency D′*
G_i *require* G_j	if $G_j = Off$ *or* 0, then $A_s = \{Off\}$ *or* $\{0\}$. Otherwise, $A_s = A$.	$A_s \cap A'_s$.
$G_i = \alpha$ *require* G_j	if $G_j = Off$ *or* 0, then $A_s = A - \{\alpha\}$. Otherwise, $A_s = A$.	$A_s \cap A'_s$.
G_i *require* $G_j = \alpha$	if $G_j \neq \alpha$, then $A_s = \{Off\}$ *or* $\{0\}$. Otherwise, $A_s = A$.	$A_s \cap A'_s$.
$G_i = \alpha_1$ *require* $G_j = \alpha_2$	if $G_j \neq \alpha_2$, then $A_s = A - \{\alpha_1\}$. Otherwise, $A_s = A$.	$A_s \cap A'_s$.
G_i *exclude* G_j	if $G_j \neq Off$ *or* 0, then $A_s = \{Off\}$ *or* $\{0\}$. Otherwise, $A_s = A$.	$A_s \cap A'_s$.
$G_i = \alpha$ *exclude* G_j	if $G_j \neq Off$ *or* 0, then $A_s = A - \{\alpha\}$. Otherwise, $A_s = A$.	$A_s \cap A'_s$.
$G_i = \alpha_1$ *exclude* $G_j = \alpha_2$	if $G_j = \alpha_2$, then $A_s = A - \{\alpha_1\}$. Otherwise, $A_s = A$.	$A_s \cap A'_s$.
G_i *at-least-one-require* G_j	if $G_j = Off$ *or* 0, then $A_s = \{Off\}$ *or* $\{0\}$. Otherwise, $A_s = A$.	if D' is (or merged from only) *at-least-one-require* that related to the same root of $\mathcal{OR}$ group as D's, then $A_s \cup A'_s$. Otherwise, $A_s \cap A'_s$.
$G_i = \alpha$ *at-least-one-require* G_j	if $G_j = Off$ *or* 0, then $A_s = A - \{\alpha\}$. Otherwise, $A_s = A$.	if D' is (or merged from only) *at-least-one-require* that related to the same root of $\mathcal{OR}$ group as that of D, then $A_s \cup A'_s$. Otherwise, $A_s \cap A'_s$.
G_i *at-least-one-exist* G_j	if $G_j = Off$ *or* 0, then $A_s = A - \{Off\}$ *or* $A - \{0\}$. Otherwise, $A_s = A$.	if D' is (or merged from only) *at-least-one-exist* that related to the same root of $\mathcal{OR}$ group as that of D, then $A_s \cup A'_s$. Otherwise, $A_s \cap A'_s$.
G_i *range-to-range* G_j (e.g., $G_i < G_j$)	if $G_j = \alpha$, then $A_s = \{\alpha_1, \ldots, \alpha_n\}$ where $\forall\alpha_n \in A$, and $\forall\alpha_n$ meets the given condition with respect to α, e.g., $\forall\alpha_n < \alpha$, etc.	$A_s \cap A'_s$.
G_i (R) *range-to* G_j	if $G_j = Off$ *or* 0, then $A_s = A - R$. Otherwise, $A_s = A$.	$A_s \cap A'_s$.
G_i (R) *range-to* $G_j = \alpha$	if $G_j \neq \alpha$, then $A_s = A - R$. Otherwise, $A_s = A$.	$A_s \cap A'_s$.
G_i *to-range* G_j (R)	if $G_j = \alpha$; $\alpha \notin R$, then $A_s = \{Off\}$ *or* $\{0\}$. Otherwise, $A_s = A$.	$A_s \cap A'_s$.
$G_i = \alpha_1$ *to-range* G_j (R)	if $G_j = \alpha_2$; $\alpha_2 \notin R$, then $A_s = A - \{\alpha_1\}$. Otherwise, $A_s = A$.	$A_s \cap A'_s$.
Additionally, when the set operation leads to an empty set, we fix $A_s = \{Off\}$ *or* $\{0\}$.		

G_i and G_j are dependent and main gene, respectively; A is the entire set of optional values for G_i; A_s denotes the set of values for G_i, given a selected value of G_j; α, α_1, α_2, and α_n denote some selected values for G_i or G_j; R is a given constrained set of range, e.g., $G_j < 10$, etc.; D' is another single or merged dependency for which G_i is the dependent gene; A'_s denotes the set of values for G_i, given the selected value(s) of the main gene(s) in D'.

—**Step 2.** If it has dependencies on different main genes, all branch nodes of one single or already combined value tree are replicated and grafted (as a whole) to each right-most branch node of another single or already combined value tree, forming new levels for the newly combined value tree representing the combinatorial conditions. Then, for the two value trees that were grafted, their leaves, whose original ancestors are now on the same path from root to a right-most node in the newly grafted tree, are combined using the set operations shown in Table 1 to create the new leaf-set.

The process stops when all related dependencies are merged and their value trees are combined, resulting in a finally combined value tree. As an example, Figure 14 illustrates the merging process

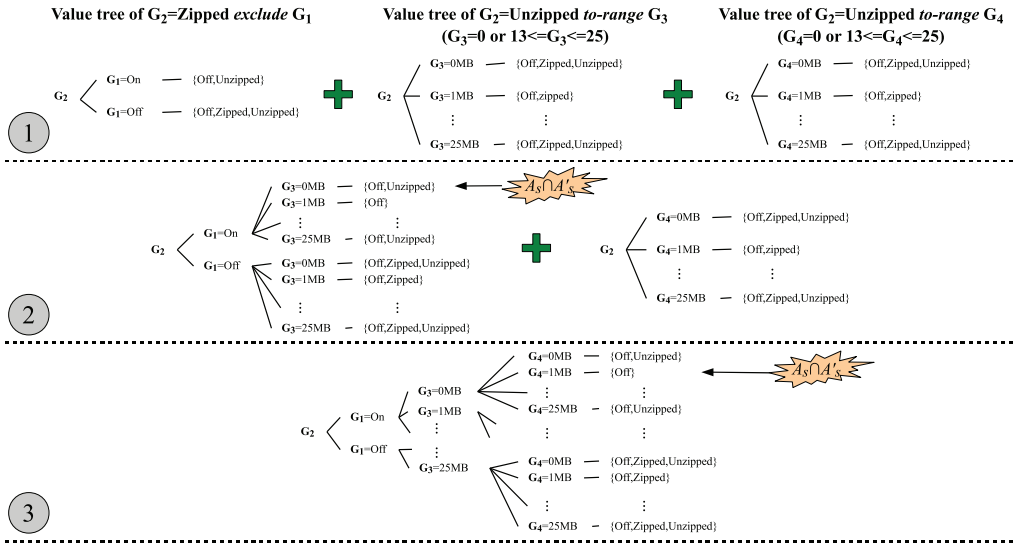


Fig. 14. The example of dependency merging and the combined value tree for the gene G_2 (*Cache Mode*), which is the dependent gene of three different main genes: G_1 (*Transmission Compression*), G_3 (*Heap Size*), and G_4 (*Disk Size*). A_s and A'_s denote two sets of values (leaf-set) to be combined, as specified in Table 1.

and the finally combined value tree for the *Cache Mode* (G_2) gene, which originally contains dependencies on three different main genes. The same merging procedure is repeated for every dependent gene.

To ensure the correctness of merging, in both steps, the dependencies that require union are always merged ahead of the others, leading to a set of partially merged dependencies. Since each of them are merged from the identical dependency type, as shown in Table 1, they can then be merged with each other (and the remaining single dependencies) using the same set operators as if they were single dependencies.

It is worth noting that, despite the fact that the given feature model is contradiction-free, it might still be possible for the combined leaves to have an empty set under some conditions. For example, recall the *Disk Size* gene from Figure 13: When its *require* dependency on *Cache Mode* merges with its *at-least-one-exist* on *Heap Size*, then combining the leaves for the condition where *Cache Mode* selects *Off* and *Heap Size* selects *0 MB* would lead to an empty set. This is due to the fact that some dependencies have different priorities: In this example, *at-least-one-exist* would constrain *Disk Size* only if it is allowed to be selected as indicated by the *require* on *Cache Mode*. In those cases, we fix {*Off*} (or {0}) as the new leaf-set for the dependent gene's combined value tree, representing the fact that it needs to be "turned off" and cannot affect the SAS under the corresponding conditions.

In Section 6, we will describe how the elitist chromosome, the extracted dependency chain(s), and the (combined) value trees of genes are seamlessly injected into the MOEA for optimizing SAS at runtime, regardless of the order of changes on genes.

6 FEATURE-GUIDED AND KNEE-DRIVEN MOEA AT RUNTIME

We have now completed all the design time transposition and analysis in FEMOSAA. Next, to optimize the SAS at runtime using FEMOSAA, we extend MOEA with the domain information gathered from the feature model, creating a feature-guided MOEA with knee selection, as shown in Figure 15 and Algorithm 2. In particular, the elitist features from Section 5 form the basic

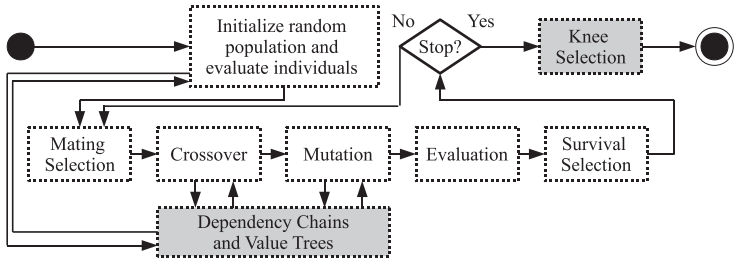


Fig. 15. The workflow of the extended feature-guided MOEA with knee selection.

ALGORITHM 2: The Extended Feature-Guided MOEA with Knee Selection

Input: The extracted dependency chain(s) C and the (combined) value trees of the genes VT, in addition to the inputs in Algorithm 1

Output: A single knee solution

```
1:  start evolution
2:  P = ∅
3:  eval = 0
4:  for i = 1 to P_size do
5:      S = getRandomSolution()
6:      doDependencyAwareMutation(S,1,C,VT)    ▷mutation rate of 1 means mutating every gene.
7:      evaluateFitness(S)
8:      eval = eval + 1
9:      P = P + S
10: end for
11: while eval < eval_max do
12:     P_0 = ∅
13:     while |P_0| ≤ P_size do
14:         parents = doMatingSelection(P)
15:         offspring = doDependencyAwareCrossover(parents,r_c,C,VT)
16:         for each solution S in offspring do
17:             doDependencyAwareMutation(S,r_m,C,VT)
18:         end for
19:         evaluateFitness(offspring)
20:         eval := eval + |offspring|
21:         P_0 = P_0 ∪ offspring
22:     end while
23:     P = P ∪ P_0
24:     doSurvivalSelection(P, P_size)
25: end while
26: P = getNonDominatedSolutions(P)
27: return getKneeSolution(P)
28: end evolution
```

chromosome representation of the solution in MOEA. To avoid exploring invalid solutions, we explicitly inject the extracted dependencies (e.g., Figure 13) and the (combined) value trees of genes (e.g., Figure 14) into the basic mutation and crossover phases of MOEA to create dependency-aware operators (Lines 6, 15, and 17). Finally, we apply the knee selection proposed in Section 6.4 to identify a single adaptation solution from the set of nondominated solutions returned by MOEA (Line 27).

6.1 Objective Functions

The elitist chromosome representation from Section 5 also helps to define the inputs of the objective (fitness) functions used in the optimization. It is worth noting that FEMOSAA works with a range of quantifiable quality objectives and is agnostic to the actual objective functions as the framework itself does not rely on any assumptions about the internal structure of those objectives. With FEMOSAA, it is also possible to consider more than two objectives but we use two in this article to provide a more intuitive illustration since the fundamental principle of multi-objective optimization is the same regardless of the objective number.

The actual objective functions exploited by FEMOSAA can be built using various modeling approaches from the literature (e.g., machine learning-based [10][14][11], analytical [44], and simulation-based [26]), as long as they are compatible with the genes identified by FEMOSAA. In Section 7.2, we elaborate the concrete objective functions, which are built by using different approaches, for each subject SAS.

6.2 Dependency-Aware Mutation Operator

At runtime, to prevent generating invalid offspring when mutating the solutions, the extracted dependency chain(s) and the (combined) value trees of the genes are seamlessly injected into the mutation operation. In this work, we use the boundary mutation operator as the basis, in which each gene might be mutated subject to a mutation rate. Upon mutation of a gene, one of its optional values is randomly assigned. The reason why the boundary mutation operator was chosen is because (i) it is one of the most commonly used mutation operators, and (ii) it works under discrete optimization problems while allowing us to randomly select a value from a predefined value tree, which particularly fits with our SAS optimization problem. However, since the violation of dependency is prevented whenever a gene is changed, it is easy to modify FEMOSAA to work with any other operators that mutate the genes in a way similar to the boundary mutation operator.

As illustrated in Algorithm 3 and Figure 16, our extended dependency-aware mutation operator has the following recursive steps:

- **Step 1.** When a gene G (e.g., *Cache Mode*) needs to be mutated, as identified by the basic mutation operator or due to its violation, we randomly select a value from its (combined) value tree with respect to the selected values of G's main genes (e.g., *Transmission Compression*, *Heap Size,* and *Disk Size*).
- **Step 2.** Then, we propagate, according to the dependency chain, to G's dependent genes (e.g., *Heap Size* and *Disk Size*), and we then validate if those genes in the solution violate any dependency using their (combined) value trees. If a violation is found, we mutate the corresponding gene and start from **Step 1** (e.g., *Heap Size = 5MB* in Figure 16), as shown in Lines 4 to 10 of the MUTATEWITHDEPENDENCY function.
- **Step 3.** The process stops when all the genes identified by the basic mutation operator have been mutated and there is no further violation found.

Using this operator, we guarantee that the mutation process in MOEA will be better guided, such that only valid adaptation solutions can be explored regardless of the order of mutation. Since the given feature model has been fully validated and exhibits no design errors, there will be at least one state such that all genes can satisfy all dependencies, which in turn prevents infinite loops in the presence of circular dependencies. As we show in Section 7.4.2, the dependency-aware operators can lead to better quality adaptations.

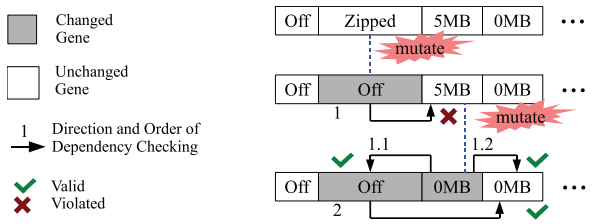


Fig. 16. Workflow of dependency-aware mutation operator. (The example genes from left to right represent *Transmission Compression*, *Cache Mode*, *Heap Size*, and *Disk Size*.)

ALGORITHM 3: Dependency-Aware Mutation Operator

Input: Given a valid solution S, the extracted dependency chain(s) C and the (combined) value trees of the genes VT

Output: A mutated valid solution (individual) S

1: **start mutation**
2: G_{set} = the genes that requires mutation as identified by the basic crossover operator based on mutation rate
3: **for each** G in G_{set} **do**
4: | MUTATEWITHDEPENDENCY(G,S)
5: **end for**
6: **return** S
7: **end mutation**

Function: MUTATEWITHDEPENDENCY(G,S)

1: VT_g = the (combined) value tree of G from VT
2: randomly *choose* a new value for G of S from VT_g
3: C_g = the chain that contains G from C
4: **for each** dependent gene of G (G_d) in C_g **do**
5: | VT_g = the (combined) value tree of G_d from VT
6: | *validate* G_d based on current conditions in VT_g
7: | **if** G_d's value in S is invalid **then**
8: | | MUTATEWITHDEPENDENCY(G_d,S)
9: | **end if**
10: **end for**

6.3 Dependency-Aware Crossover Operator

Similar to the mutation process, it is necessary to eliminate invalid offspring when swapping elements of the solutions. To this end, the extracted dependency chain(s) and the (combined) value trees can be injected into the given crossover operator[7] in the MOEA. In this work, we rely on the widely used uniform crossover, where two genes, each from a different parent and both at the same position in the chromosome, might be swapped subject to a crossover rate. Such a uniform crossover operator was chosen because it mitigates the problem of gene locus; that is, the ability to explore the search space is less sensitive to the closeness of highly dependent genes (features) in the encoding, which helps to relax extra design requirements of the SAS. However, since the violation of dependency is prevented whenever a pair of genes is swapped, it is easy to modify

[7]In this work, we used the most common type of crossover operators, one that takes two parents and produces two offspring. However, our dependency can be injected with other types of crossover operators (which are rare) in a similar way.

FEMOSAA to work with any other operators in which each pair of the swapped genes would be always at the same position in the encoding.

After the injection, as shown in Algorithm 4 and Figure 17, our dependency-aware crossover operator uses the following recursive steps:

- **Step 1.** When a gene G (e.g., *Disk Size*) needs to be swapped, as identified by the basic crossover operator or due to violation, we swap it in the offspring if it has not been swapped already.
- **Step 2.** We propagate, according to the dependency chain, to G's dependent genes (*Heap Size* and *Cache Mode*). If a dependent gene in an offspring violates the dependency, we then attempt to swap the gene by repeating from **Step 1** (e.g., *Heap Size = 0MB* in Figure 17), as shown in Lines 7 to 13 of the CROSSOVERWITHDEPENDENCY function.
- **Step 3.** Next, we check if G in an offspring violates dependencies; if a violation exists, we then attempt to swap all the main genes of G (e.g., *Cache Mode* and *Heap Size*, although this is not needed in the example shown) by repeating from **Step 1** for each of them, as shown in Lines 14 to 20 of the CROSSOVERWITHDEPENDENCY function.
- **Step 4.** The process terminates when there is no dependency violation or all genes in the parents have been swapped.

In this way, we guarantee that only valid adaptation solutions are produced, given any order of crossover. As we show in Section 7.4.2, dependency-aware operators can lead to better quality adaptations.

6.4 Knee Selection

As mentioned in Section 3.2, knee points are those solutions that achieve well-balanced tradeoff on all objectives. This is particularly appealing as FEMOSAA aims for those cases where the relative importance between conflicting objectives of SAS is unknown and it is too difficult to quantify them. As we can see in Figure 18, which shows solutions that minimize both response time and energy consumption, those more balanced knee solutions are likely to occur around the visible bulge, representing a good sense of compromise. Intuitively, the knee solutions tend to be the furthest away from all solutions that have the worst result on each single objective (i.e., the extreme solutions). As a result, finding the knee solutions within a nondominated set is equivalent to searching for the solution(s) that has the largest general distance from extreme solutions in the set.

To achieve this, we developed a knee selection method for selecting a single solution from the set returned by MOEA. As shown in Figure 18, given the final nondominated set obtained by the feature-guided MOEA, we at first construct a line ℓ that connects the extreme solutions holding the worst value at each single objective. Then, we calculate the perpendicular distance from each nondominated solution $\mathbf{x}$ to the ℓ:

$$d(\mathbf{x}, \ell) = \begin{cases} \frac{|\epsilon|}{\sqrt{a^2+b^2}}, & if\ \epsilon < 0 \\ -\frac{|\epsilon|}{\sqrt{a^2+b^2}}, & otherwise, \end{cases} \tag{1}$$

where $\epsilon = a \times f(\mathbf{x}) + b \times g(\mathbf{x}) + c$ and the parameters a, b, and c can be identified through the extreme solutions. In particular, $\epsilon < 0$ means that $\mathbf{x}$ is on the left side of ℓ; otherwise, $\mathbf{x}$ is on the right side of ℓ. Clearly, solutions on the left side of ℓ are preferable to those on the right when considering a minimization problem. The solution(s) that has the largest perpendicular distance to ℓ is the knee solution(s) that we are seeking. When there are multiple knee solutions, we randomly select one for adaptation. As we show in Section 7.4.3, the knee selection can lead to more balanced

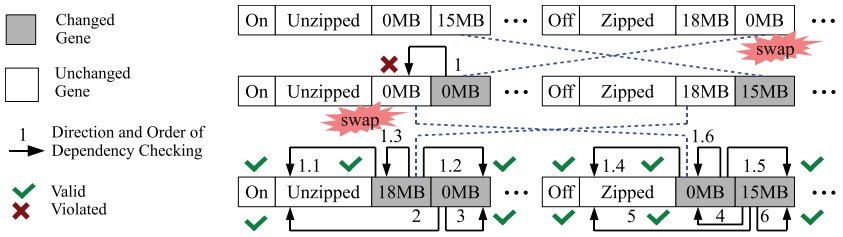


Fig. 17. Workflow of dependency-aware crossover operator. (The example genes from left to right represent *Transmission Compression*, *Cache Mode*, *Heap Size*, and *Disk Size*.)

ALGORITHM 4 : Dependency-Aware Crossover Operator

Input: Given two valid parent solutions S_1 and S_2, the extracted dependency chain(s) C and the (combined) value trees of the genes VT

Output: Two new valid solutions (individuals) S_{n1} and S_{n2}

start crossover
$S_{n1} = S_1$
$S_{n2} = S_2$
G_{set} = the genes that requires crossover as identified by the basic crossover operator based on crossover rate
for each G in G_{set} **do**
  crossoverWithDependency(G,S_1,S_2,S_{n1},S_{n2})
end for
return S_{n1} and S_{n2}
end crossover

Function: crossoverWithDependency(G,S_1,S_2,S_{n1},S_{n2})

if G in S_{n1} and S_{n2} has already been swapped **then**
  return
end if
swap the values of G in S_{n1} and S_{n2}
C_g = the chain that contains G from C
for each solution S in {S_{n1},S_{n2}} **do**
  for each dependent gene of G (G_d) in C_g **do**
    VT_g = the (combined) value tree of G_d from VT
    validate G_d based on current conditions in VT_g
    if G_d's value in S is invalid **then**
      crossoverWithDependency(G_d,S_1,S_2,S_{n1},S_{n2})
    end if
  end for
  VT_g = the (combined) value tree of G from VT
  validate G based on current conditions in VT_g
  if G's value in S is invalid **then**
    for each main gene of G (G_m) in C_g **do**
      crossoverWithDependency(G_m,S_1,S_2,S_{n1},S_{n2})
    end for
  end if
end for

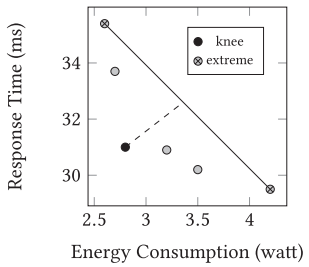


Fig. 18. Finding knee solution(s) in the nondominated set.

tradeoffs on the quality of adaptations. Note that by replacing ℓ with a surface or a hyperplane, we can easily extend our knee selection to more than two objectives.

6.5 The Concrete MOEAs

Without loss of generality, FEMOSAA can work easily with a wide range of MOEAs. In this work, we run FEMOSAA with three distinct MOEAs (i.e., MOEA/D-STM, NSGA-II, and IBEA), each of which is a widely used representative of its own category, as explained in Section 3.1. In the following, we briefly explain their principles and ideas, although the details are beyond the scope of this article.

- NSGA-II [20]—As one of the most popular MOEAs, NSGA-II first uses nondominated sorting to divide the population into several nondomination levels. Solutions in the first several levels have a higher priority to survive to the next iteration. If the size of the current nondominated solution set exceeds the predefined threshold, NSGA-II uses the crowding distance, a density estimation technique, to trim the population.
- IBEA [53]—The basic idea of IBEA is to first define the optimization goal in terms of a binary performance measure/indicator, which is then used to guide the survival selection process. In this way, IBEA transfers a multi-objective optimization problem into a new single-objective optimization problem, with respect to the chosen indicator, to facilitate the fitness assignment procedure.
- MOEA/D-STM [36]—MOEA/D is a MOEA framework that combines the mathematical rigour of the classic multi-objective optimization method and the implicit parallelism of evolutionary algorithms in a single paradigm. Different from the classic multi-objective optimization method [38], which can only obtain a single Pareto-optimal solution at a time by aggregating all objectives into a single-objective aggregation function, MOEA/D decomposes the original multi-objective optimization problem into a population of single-objective optimization subproblems. In particular, each subproblem corresponds to a predefined weight vector generated in a systematic manner [35]. Afterwards, MOEA/D uses a population-based technique to solve these subproblems in a collaborative manner. As a recent variant of MOEA/D, MOEA/D-STM achieves a balance between convergence and diversity by modifying the survival selection mechanism of the original MOEA/D. In a nutshell, MOEA/D-STM treats subproblems and solutions as two sets of agents. Each agent has its preferences over the agents on the other side. In particular, subproblems concern convergence, while solutions concern diversity. The survival selection process is modelled as a matching process between subproblems and solutions. The stable matching between them finally turns out to be the selection result. Note that the stable matching achieves an equilibrium between the preferences of subproblems and solutions; thus, the selection strikes a balance between convergence and diversity simultaneously.

7 EXPERIMENTS AND EVALUATION

By using the actual running SAS that consists of a stack of real-world software, we conducted comprehensive experiments to evaluate the effectiveness of FEMOSAA by comparing it with its variants and state-of-the-art frameworks under different metrics. The source code of FEMOSAA, the comparative variants and state-of-the-art frameworks, the subject SAS, and all the experiment data can be publicly accessed via GitHub.[8,9] Specifically, our evaluation aims to answer the following research questions:

- **RQ1.** What are the added values of the elitist chromosome representation in contrast to the conventional binary representation?
- **RQ2.** What benefits do the dependency-aware operators provide compared to classical and widely used operators?
- **RQ3.** What benefits does the knee selection mechanism provide in contrast to selecting arbitrary nondominated solutions?
- **RQ4.** What is the effectiveness of FEMOSAA in contrast to other state-of-the-art search-based frameworks?
- **RQ5.** What is the overhead of FEMOSAA, in terms of execution time, compared to other state-of-the-art frameworks? Which part(s) cause the most overhead? Is the overhead suitable for SAS runtime?

7.1 Verifiability and Methodology

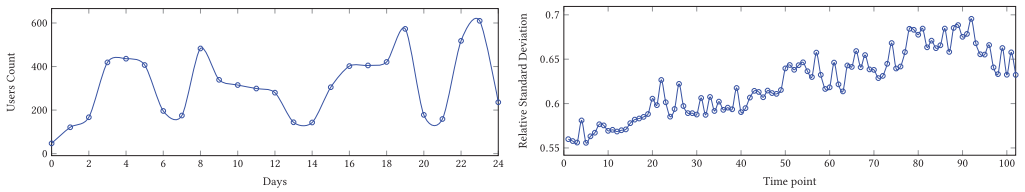
We conducted experiments on a dedicated server that runs Ubuntu Linux 14.04 on an Intel i5 2.8GHz Quad Core processor, 4GB RAM. To separate the adaptation engine and the adaptable software, we used Xen v3.0.3 [48] as the hypervisor to create a virtualized environment. We implemented FEMOSAA as the adaptation engine using Java, JDK 1.6, and it is deployed on the *Dom0* of Xen. To set up all the MOEAs, we use a population size of 100 for 10 generations as the termination criteria; the mutation and crossover rates are 0.1 and 0.9, respectively. In particular, for IBEA, we use the ϵ-indicator and an archive size of 500; whereas for MOEA/D-STM, we apply Tchebycheff aggregation to create subproblems where the number of evenly distributed weight vectors is 100 (i.e., 100 subproblems) and the size of each subproblem's neighborhood is 20. Those settings are either common values or have been tailored for runtime optimization in our cases with respect to quality and overhead. All MOEAs used in the experiments are extended from the jMetal Framework [21]. To mitigate interference caused by the adaptation engine, we used one vCPU and 800MB RAM on *Dom0*.

7.2 The Subject SAS

We deploy two running SAS for our real-time experiments. The two diverse subject SAS aims to examine the generality and applicability of FEMOSAA under different domains. On the micro level, they help to demonstrate how FEMOSAA can be applied to different feature models, dependency structures, environmental factors, dimensions of quality objectives, and degrees of objective conflicts:

—***RUBiS-SAS.*** This adaptable software is a software stack that contains RUBiS [43], which is a well-known software benchmark that simulates the *eBay* model, and a set of real-world software including Tomcat v6.0.28 [2], MySQL v3.23.58 [40], and Linux kernel v2.6.16.29

[8]https://github.com/taochen/ssase.
[9]https://github.com/JerryI00/Software-Adaptive-System.



(a) The compressed FIFA98 workload for all patterns in *RUBiS-SAS* (from June 7 to July 1, 1998)

(b) The mean relative standard deviation of all concrete services' throughput and cost in *SOA-SAS*

Fig. 19. The changing environment of a subject SAS.

running on a configurable guest virtual machine. Ehcache v2.6 [3] is plugged to RUBiS as the cache management module. All the features that can be adapted at runtime are represented by the feature model in Figure 4, with a total of 1,151 features and a search space of around 1.3×10^{16} using the elitist chromosome representation (including both valid and invalid solutions). We also use two distinct workload patterns, a read-write patten and a read-only one, to create diverse runtime behaviors in the SAS. To simulate realistic time-varying environmental conditions within the capacity of our testbed, we vary the number of clients according to the compressed FIFA98 workload trace [5], as shown in Figure 19(a). This setup can generate up to 600 parallel requests, which offers sufficient dynamism and uncertainty. The workload is generated by another machine using the client emulator provided in RUBiS. In general, a heavier workload implies more pressure on the SAS, which reduces the number of effective solutions and thus makes the problem harder. The goal is to continually optimize the following two conflicting quality attributes that need to be minimized:

(1) ***Response Time:*** In the *RUBiS-SAS*, response time is measured as the elapsed time between a request's arrival and its response. To express the quality of *RUBiS-SAS* by the end of a time point, we calculated the average response time for all monitored requests served in the past time interval.

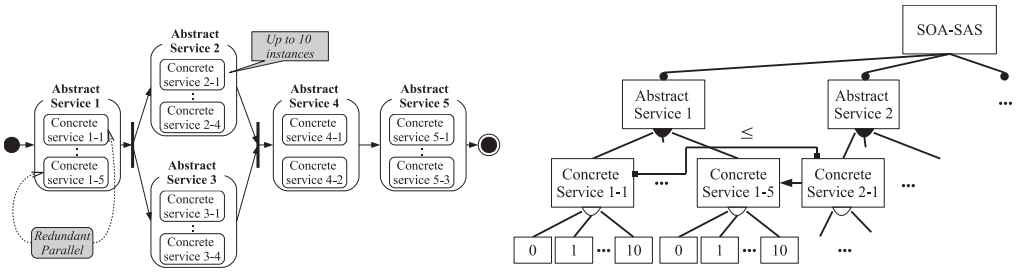
(2) ***Energy Consumption:*** The energy consumed by software systems, which accounts for 2% of the global carbon emissions in 2007 [39], is increasingly becoming an important quality concern that has clear conflicts with response time. To measure energy consumption, we leveraged PowerAPI [7], a tool that measures the actual energy (watt) incurred by the software's CPU and memory utilization through probing into the sensors of the hardware infrastructure. For each time interval, we computed the average of 30 measurement results from PowerAPI as the energy consumed by the *RUBiS-SAS* for that interval.

These two quality attributes serve as a case with moderate degree of conflict. In other words, it is easier to find solutions that have better response time and energy consumption when the workload is lower. However, as the workload increases, the degree of conflict between them tends to amplify. Formally, the objective functions for *RUBiS-SAS*'s quality attributes are defined as:

$$RT(t+1) = f(G_1(t+1), G_2(t+1), \ldots, G_n(t+1), \delta(t)) \tag{2}$$

$$EC(t+1) = g(G_1(t+1), G_2(t+1), \ldots, G_n(t+1), \delta(t)), \tag{3}$$

whereby $G_1(t+1), G_2(t+1), \ldots, G_n(t+1)$ denote the genes and their selected values in a possible adaptation solution, and $\delta(t)$ denotes a set of most recent environmental factors



(a) Composition workflow (b) A fragment of the feature model

Fig. 20. The architecture of *SOA-SAS*.

(e.g., workload in this article). Given an adaptation solution, $RT(t+1)$ and $EC(t+1)$ are the expected fitness values for response time and energy consumption, respectively.

In *RUBiS-SAS*, we adopt the machine learning-based model from [10][14][11] to build the objective functions for *RUBiS-SAS* since it relies on few assumptions of the application domain. By using the actual data of quality performance, software status, and environment monitored from the SAS, this model can be continually updated at runtime to provide sufficient accuracy [11]. To stabilize the objective functions in this work, we pretrained these models by monitoring the data from SAS under random workload for 120 intervals, after which the models are gradually and dynamically updated at runtime. This step is similar to the testing phase occurring before actual production deployment in the industry. As we show for the next subject SAS, other modeling approaches (e.g., analytical [44] or simulation-based [26]) can be easily applied as long as they are compatible with the genes identified by FEMOSAA.

To emulate the behaviors of a running software system, we run the SAS, empowered with FEMOSAA, under the entire FIFA98 workload trace, where the sampling interval is 120s for a total of 102 timesteps, leading to around 5 hours per experiment run, including the emulated end-users' thinking time. For all experiments, we trigger an optimization run by the end of each interval.

—***SOA-SAS.*** This is a Service-Oriented Architecture (SOA)-based adaptable software derived from Wada et al. [49]. At the highest level, it is composed of five abstract services connected in parallel or in sequence, as shown in Figure 20. Each abstract service can select 2 to 5 concrete services, which are redundantly parallel and associated with different quality values. Each concrete service could have up to 10 concurrent instances with the same quality of throughput and cost. Those abstract services, concrete services, and their replicas are all considered as features of *SOA-SAS* (e.g., a fragment in Figure 20), creating a total of 221 features in the feature model with a search space of around 5.6×10^{18} using the elitist chromosome representation (including both valid and invalid solutions). We also placed various categorical and numeric dependencies on the feature model. To simulate dynamism and uncertainty under time-varying environmental conditions, we amended the throughput and cost of certain concrete services at each timestep by changing their diversity level according to Gaussion distribution, as shown in Figure 19(b). Generally, more diverse concrete services imply more sparse adaptation solutions, which reduces the number of effective solutions and thus makes the problem harder. Here, we consider the following conflicting objectives to be maximized and minimized for the entire composition:

(1) ***Throughput:*** In the *SOA-SAS*, throughput of each concrete service is its maximum capacity under normal operation, expressed as number of requests per second. The throughput of the entire composition is calculated via an aggregate analytical model, as shown in Equation (4).
(2) ***Cost:*** Each concrete service, when utilized, comes with a monetary cost. The cost of the entire composition is again calculated via an aggregate analytical model, as shown in Equation (5).

As suggested in Wada et al. [49], the basic throughput and cost for each concrete service in *SOA-SAS* were specified following two different Gaussian distributions such that a concrete service with higher throughput would have higher cost, too. In *SOA-SAS*, we leverage an analytical model to define the objective functions:

$$T(t+1) = \min\left\{a \in A : \sum_{i=1}^{a} G_i(t+1) \times T_i(t)\right\} \tag{4}$$

$$C(t+1) = \sum_{i=1}^{n} G_i(t+1) \times C_i(t), \tag{5}$$

where $G_i(t+1)$ is the ith gene (a concrete service) and its selected value (the actual number of instances for that concrete service) in a possible adaptation solution for the total of n genes. $T_i(t)$ and $C_i(t)$ are the related throughout and cost value, respectively. a refers to the genes associated with a given abstract service, denoted by A. Since the analytical model is customizable, we forced the objective functions to produce a worst possible fitness value for any invalid solutions, creating a stronger pressure for them to be eliminated in the classic, non-dependency aware approaches.

Similar to *RUBiS-SAS*, there is a total of 102 timesteps, and we trigger an optimization run by the end of each interval.

As discussed, FEMOSAA finds elitist features and extracts their dependencies at design time, after which the outcomes are passed to the MOEA and knee selection for runtime optimization.

7.3 The Metrics

We consider the following metrics to evaluate various aspects of FEMOSAA:

— **Individual Quality Attribute:** In addition to the detailed plots of the results, we also report on the Geometric Mean (GM) for all the observed objective values over 102 intervals, as GM tends to be more resilient to the outliers than the arithmetic mean even though it still can be influenced by those outliers. This is important because outliers on the achieved quality are common in real-world scenarios; they cannot be eliminated, but we need to prevent them from strongly dominating the overall result.
— **Aggregate Quality of SAS:** To compare the overall quality for both objectives and provide an overall assessment, we apply a modified Hypervolume (HV) [55][37] for two objectives and Euclidean Distance (ED) [50] on GMs to assess the balance in tradeoffs and the extent to which both objectives are optimized, respectively. The GMs are normalized before computing those metrics so that they range from 0 to 1. Hence, the modified HV is computed as:

$$HV = \prod_{i=1}^{n}\left(1 - \frac{GM_i - GM_{i,b}}{GM_{i,w} - GM_{i,b}}\right) \tag{6}$$

and ED can be calculated by:

$$ED = \frac{1}{n} \times \sqrt{\sum_{i=1}^{n} \left(\frac{GM_i - GM_{i,b}}{GM_{i,w} - GM_{i,b}} - 0 \right)^2}, \tag{7}$$

where n is the number of objectives and GM_i is the GM for the ith objective; $GM_{i,b}$ and $GM_{i,w}$ are the best and worst GMs that we observed for the ith objective, respectively. Notably, we converted the maximizing objective (e.g., throughput) into a minimizing one by inverting the results.

— **Percentage of Valid Solutions Found:** For all the intervals, we also compare the average percentage of valid adaptation solutions found in the final population.

— **Running Overhead:** We report on the mean running overhead, in terms of the execution time, over all the intervals in the experiment runs.

To confirm the statistical significance of the comparisons on the quality attributes, we performed a *Wilcoxon Signed-Rank* test (two-tailed) for all comparisons between FEMOSAA (or FEMOSAA-N) and the others as our data do not follow Gaussian distributions. We use 95% as the confidence interval ($\alpha = 0.05$), which means that if the test produces a p smaller than 0.05, we can reject the null hypothesis H_0, which states that the given two approaches cannot be statistically distinguished when optimizing a quality attribute of the SAS. The effect size for each test is also reported, and we follow the categories in Kampenes et al. [32] to measure the meaningfulness of effect size.

7.4 Effectiveness of FEMOSAA

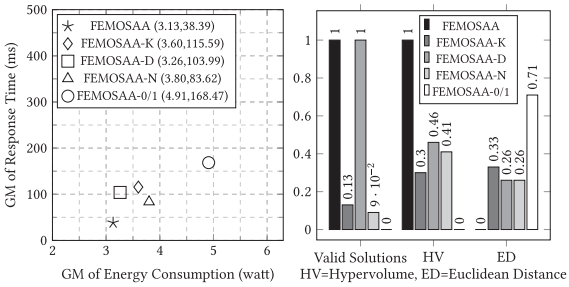
To ensure generality of the evaluation, we run each of the three MOEAs for optimizing the SAS under each case, which results in a total of nine scenarios. Further, to evaluate the effectiveness of FEMOSAA in each scenario, we compare FEMOSAA with a number of its variants:

FEMOSAA-K—This is similar to our FEMOSAA except that it relies on the classic, non-dependency aware operators. Therefore, if the final population contains invalid solutions, it automatically filters them and works only on the valid ones. When no valid solutions are found in the final nondominated set, it corrects the invalid solutions via a dependency-aware mutation operator. FEMOSAA-K aims to examine whether the specifically tailored dependency-aware operators can achieve benefit over the classic operators that are widely used in SBSE.

FEMOSAA-D—This variant of FEMOSAA is without knee selection. Hence, in the final nondominated set, one solution is randomly selected for adaptation. FEMOSAA-D aims to examine the importance of considering knee in runtime SAS optimization. Notably, the use of a randomized baseline is strongly recommended by the existing SBSE community [4].

FEMOSAA-N—This variant considers neither dependency nor knee selection in the evolutionary optimization. Hence, it uses the same ad hoc strategies from FEMOSAA-D and FEMOSAA-K. FEMOSAA-N is designed to evaluate the combinatorial benefit of dependency-aware operators and the selection.

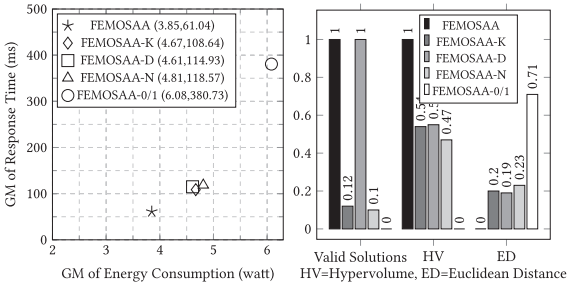
FEMOSAA-0/1—A baseline variant, built from FEMOSAA-N, this exploits the conventional binary chromosome representation using all the features from the feature model. Thus, it works under a search space of 2^{1151} and 2^{221} for the two-subject SAS, as opposed to the search space of 1.3×10^{16} and 5.6×10^{18} when using the elitist chromosome representations. FEMOSAA-0/1 considers neither dependency nor knee in the evolutionary optimization. Note that if more than one members in a $\mathcal{XOR}$ group is selected, we then randomly choose one among them to create a computable solution for the fitness functions. FEMOSAA-0/1 aims to evaluate whether the elitist chromosome representation outperforms the commonly used binary one.



p value (effect size) on Response Time		
	FEMOSAA	FEMOSAA-N
FEMOSAA-K	**.002 (medium)**	.384 (trivial)
FEMOSAA-D	**.003 (medium)**	.638 (trivial)
FEMOSAA-N	**.030 (medium)**	—
FEMOSAA-0/1	**<.001 (large)**	**.042 (small)**

p value (effect size) on Energy Consumption		
	FEMOSAA	FEMOSAA-N
FEMOSAA-K	.153 (small)	.516 (trivial)
FEMOSAA-D	.535 (trivial)	.051 (trivial)
FEMOSAA-N	**.002 (medium)**	—
FEMOSAA-0/1	**<.001 (large)**	**.018 (medium)**

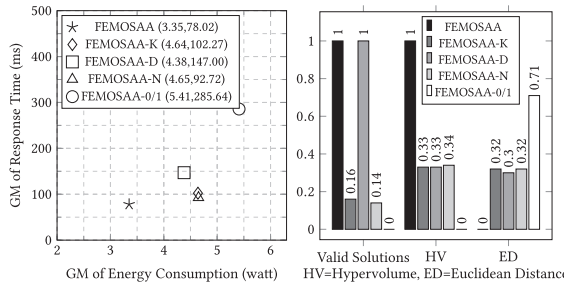
Fig. 21. Results with MOEA/D-STM under read-write pattern on RUBiS-SAS. (GM denotes Geometric Mean. The significant statistics of comparisons, i.e., $p < 0.05$, are highlighted and shown in bold.)



p value (effect size) on Response Time		
	FEMOSAA	FEMOSAA-N
FEMOSAA-K	**.046 (small)**	.960 (trivial)
FEMOSAA-D	**.025 (medium)**	.968 (trivial)
FEMOSAA-N	**.043 (small)**	—
FEMOSAA-0/1	**<.001 (large)**	**.003 (large)**

p value (effect size) on Energy Consumption		
	FEMOSAA	FEMOSAA-N
FEMOSAA-K	**.034 (medium)**	.810 (trivial)
FEMOSAA-D	.110 (small)	.562 (trivial)
FEMOSAA-N	**.032 (medium)**	—
FEMOSAA-0/1	**<.001 (large)**	**.019 (large)**

Fig. 22. Results with NSGA-II under read-write pattern on RUBiS-SAS. (GM denotes Geometric Mean. The significant statistics of comparisons, i.e., $p < 0.05$, are highlighted and shown in bold.)



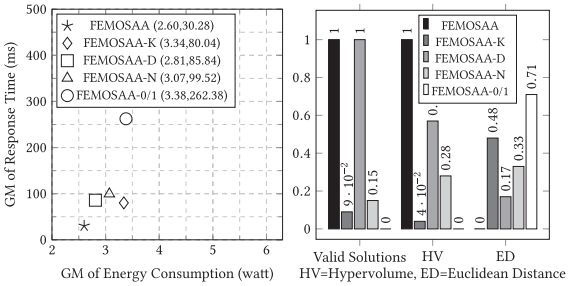
p value (effect size) on Response Time		
	FEMOSAA	FEMOSAA-N
FEMOSAA-K	**.041 (small)**	.424 (trivial)
FEMOSAA-D	**.015 (medium)**	.211 (small)
FEMOSAA-N	.459 (trivial)	—
FEMOSAA-0/1	**<.001 (large)**	**.001 (large)**

p value (effect size) on Energy Consumption		
	FEMOSAA	FEMOSAA-N
FEMOSAA-K	**<.001 (large)**	.952 (trivial)
FEMOSAA-D	**.023 (medium)**	.407 (trivial)
FEMOSAA-N	**<.001 (large)**	—
FEMOSAA-0/1	**<.001 (large)**	**.038 (medium)**

Fig. 23. Results with IBEA under read-write pattern on RUBiS-SAS. (GM denotes Geometric Mean. The significant statistics of comparisons, i.e., $p < 0.05$, are highlighted and shown in bold.)

We report the results for all cases from Figures 21 to 29, and we also plot the achieved quality for all timesteps with respect to the environmental conditions in Figures 30 and 31, as well as the results of an example optimization run in Figure 32. Note that, in Figures 30 and 31, the area near the bottom-left line of the cube is the ideal area; the closer the points converge to that line, the better the overall result.

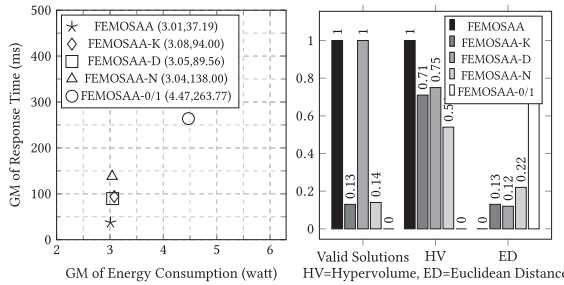
7.4.1 Evaluating the Elitist Chromosome Representation. To evaluate the effectiveness of our elitist chromosome representation in contrast to the conventional binary representation, we first compare FEMOSAA with FEMOSAA-0/1 for all cases on *RUBiS-SAS*, as shown in Figures 21 to 26. Clearly, we see that FEMOSAA greatly outperform FEMOSAA-0/1 on both response time and



p value (effect size) on Response Time		
	FEMOSAA	FEMOSAA-N
FEMOSAA-K	**.010 (medium)**	.472 (trivial)
FEMOSAA-D	**.043 (small)**	.681 (small)
FEMOSAA-N	**.015 (trivial)**	—
FEMOSAA-0/1	**<.001 (large)**	**.005 (large)**

p value (effect size) on Energy Consumption		
	FEMOSAA	FEMOSAA-N
FEMOSAA-K	**.017 (medium)**	.603 (trivial)
FEMOSAA-D	.787 (trivial)	.126 (small)
FEMOSAA-N	**<.049 (small)**	—
FEMOSAA-0/1	**<.049 (small)**	.795 (trivial)

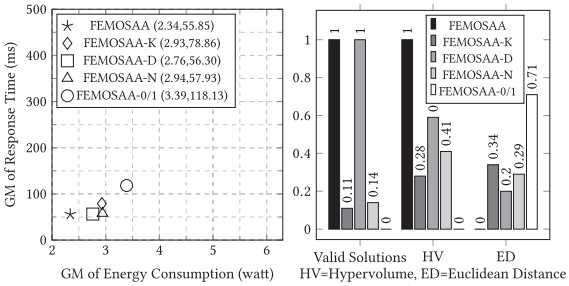
Fig. 24. Results with MOEA/D-STM under read-only pattern on RUBiS-SAS. (GM denotes Geometric Mean. The significant statistics of comparisons, i.e., $p < 0.05$, are highlighted and shown in bold.)



p value (effect size) on Response Time		
	FEMOSAA	FEMOSAA-N
FEMOSAA-K	**.010 (medium)**	.242 (small)
FEMOSAA-D	**.004 (medium)**	.549 (trivial)
FEMOSAA-N	**<.001 (large)**	—
FEMOSAA-0/1	**<.001 (large)**	.080 (medium)

p value (effect size) on Energy Consumption		
	FEMOSAA	FEMOSAA-N
FEMOSAA-K	.968 (trivial)	.904 (trivial)
FEMOSAA-D	.992 (trivial)	.697 (trivial)
FEMOSAA-N	.624 (trivial)	—
FEMOSAA-0/1	**.002 (medium)**	**.002 (medium)**

Fig. 25. Results with NSGA-II under read-only pattern on RUBiS-SAS. (GM denotes Geometric Mean. The significant statistics of comparisons, i.e., $p < 0.05$, are highlighted and shown in bold.)



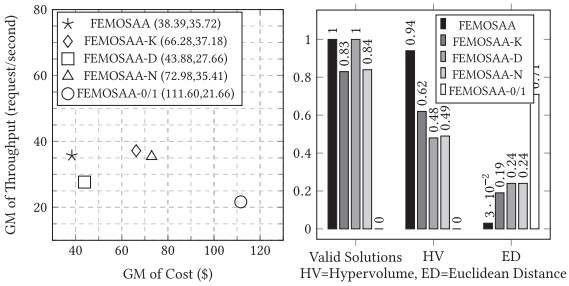
p value (effect size) on Response Time		
	FEMOSAA	FEMOSAA-N
FEMOSAA-K	.313 (small)	.549 (trivial)
FEMOSAA-D	.174 (small)	.596 (trivial)
FEMOSAA-N	.490 (small)	—
FEMOSAA-0/1	**.028 (small)**	**.047 (medium)**

p value (effect size) on Energy Consumption		
	FEMOSAA	FEMOSAA-N
FEMOSAA-K	**.007 (medium)**	.881 (trivial)
FEMOSAA-D	.535 (trivial)	.624 (trivial)
FEMOSAA-N	**.006 (medium)**	—
FEMOSAA-0/1	**.005 (medium)**	.952 (trivial)

Fig. 26. Results with IBEA under read-only pattern on RUBiS-SAS. (GM denotes Geometric Mean. The significant statistics of comparisons, i.e., $p < 0.05$, are highlighted and shown in bold.)

energy consumption with $p < 0.05$ and nontrivial effect sizes. Further, FEMOSAA yields better HV and ED values for all cases. In Figure 30, we also note that, in contrast to FEMOSAA-0/1 under all cases, the quality achieved by FEMOSAA tends to be much more convergent to the left-bottom line of the cube even under heavy workload, meaning that it leads to better quality results and a more balanced tradeoff. For *SOA-SAS*, we observe similar results on throughput and cost, as shown in Figures 27 to 29 and Figure 31.

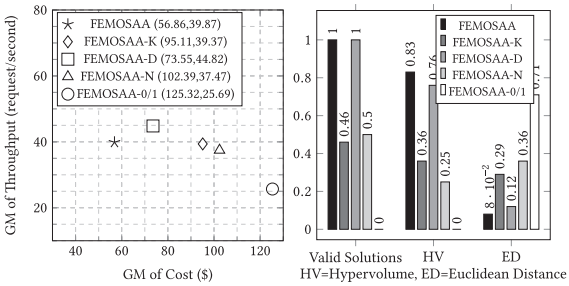
However, the preceding comparison only demonstrates the combinatorial effectiveness of elitist chromosome representation, dependency-aware operators, and knee selection, while it is not clear whether superiority over the conventional binary representation truly results from the elitist chromosome representation. To answer this, we then compare FEMOSAA-N with FEMOSAA-0/1



p value (effect size) on Throughput		
	FEMOSAA	FEMOSAA-N
FEMOSAA-K	.596 (trivial)	.463 (trivial)
FEMOSAA-D	**.033 (medium)**	.182 (small)
FEMOSAA-N	.704 (trivial)	—
FEMOSAA-0/1	**<.001 (large)**	**<.001 (large)**

p value (effect size) on Cost		
	FEMOSAA	FEMOSAA-N
FEMOSAA-K	**<.001 (large)**	.097 (small)
FEMOSAA-D	**.048 (small)**	**<.001 (large)**
FEMOSAA-N	**<.001 (large)**	—
FEMOSAA-0/1	**<.001 (large)**	**<.001 (large)**

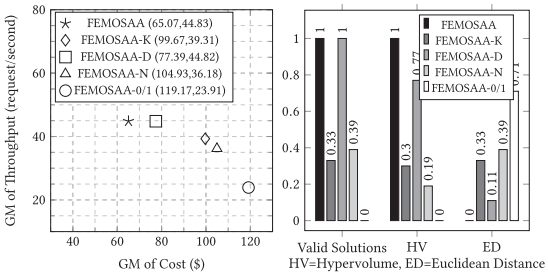
Fig. 27. Results with MOEA/D-STM on SOA-SAS. (GM denotes Geometric Mean. The significant statistics of comparisons, i.e., $p < 0.05$, are highlighted and shown in bold.)



p value (effect size) on Throughput		
	FEMOSAA	FEMOSAA-N
FEMOSAA-K	.352 (small)	.831 (trivial)
FEMOSAA-D	**.016 (medium)**	**.002 (medium)**
FEMOSAA-N	.300 (small)	—
FEMOSAA-0/1	**<.001 (large)**	**<.001 (large)**

p value (effect size) on Cost		
	FEMOSAA	FEMOSAA-N
FEMOSAA-K	**<.001 (large)**	**.015 (medium)**
FEMOSAA-D	**<.001 (large)**	**<.001 (large)**
FEMOSAA-N	**<.001 (large)**	—
FEMOSAA-0/1	**<.001 (large)**	**<.001 (large)**

Fig. 28. Results with NSGA-II on SOA-SAS. (GM denotes Geometric Mean. The significant statistics of comparisons, i.e., $p < 0.05$, are highlighted and shown in bold.)

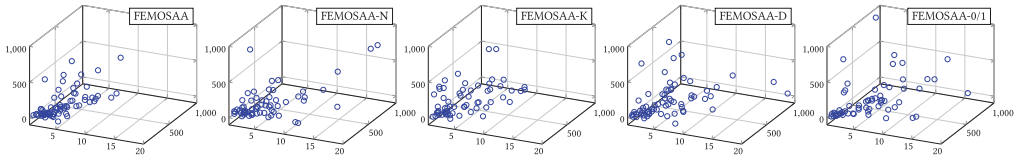


p value (effect size) on Throughput		
	FEMOSAA	FEMOSAA-N
FEMOSAA-K	**<.001 (large)**	.515 (trivial)
FEMOSAA-D	.524 (trivial)	**.004 (medium)**
FEMOSAA-N	**<.001 (large)**	—
FEMOSAA-0/1	**<.001 (large)**	**<.001 (large)**

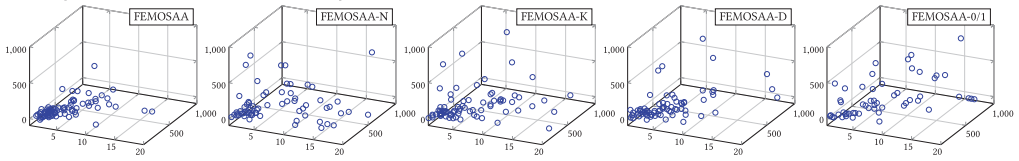
p value (effect size) on Cost		
	FEMOSAA	FEMOSAA-N
FEMOSAA-K	**<.001 (large)**	.055 (small)
FEMOSAA-D	**<.001 (large)**	**<.001 (large)**
FEMOSAA-N	**<.001 (large)**	—
FEMOSAA-0/1	**<.001 (large)**	**<.001 (large)**

Fig. 29. Results with IBEA on SOA-SAS. (GM denotes Geometric Mean. The significant statistics of comparisons, i.e., $p < 0.05$, are highlighted and shown in bold.)

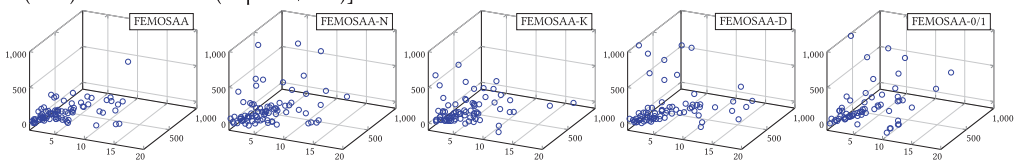
for all cases on *RUBiS-SAS*, as shown in Figures 21 to 26. We see that, again, FEMOSAA-N yields much better results on both quality attributes than FEMOSAA-0/1 under all cases, while such improvements are statistically significant on at least one attribute with nontrivial effect sizes. It also achieves better HV and ED results. Further, although it is less than 15%, FEMOSAA-N does produce more valid solutions than FEMOSAA-0/1, which cannot identify any valid solution and thus affect the quality of optimization. Next, we take a more detailed comparison between FEMOSAA-N and FEMOSAA-0/1 through Figure 30, which reveals that, for all cases, the results of FEMOSAA-N are closer to the left-bottom line of the cube when the workload is low (e.g., less than around 500 requests/min). For *SOA-SAS*, we observe even better results: FEMOSAA-N largely outperforms FEMOSAA-0/1 on all metrics, and the comparisons on quality attributes exhibit $p < 0.05$ and large



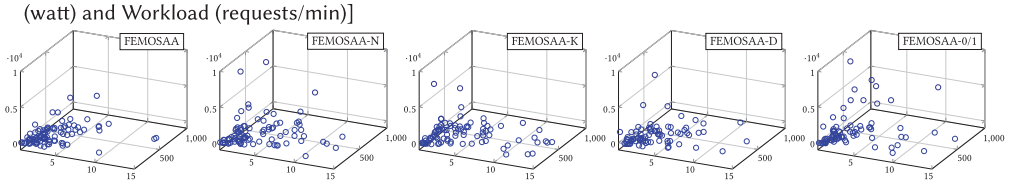
(a) MOEA/D-STM and read-write pattern [the axis from left to right are Response Time (ms), Energy Consumption (watt) and Workload (requests/min)]



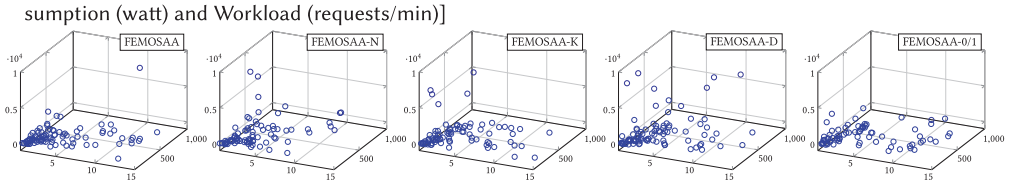
(b) NSGA-II and read-write pattern [the axis from left to right are Response Time (ms), Energy Consumption (watt) and Workload (requests/min)]



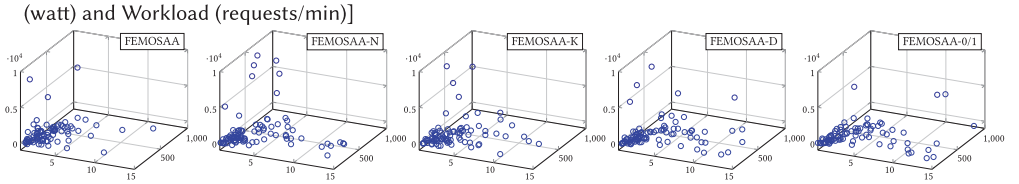
(c) IBEA and read-write pattern [the axis from left to right are Response Time (ms), Energy Consumption (watt) and Workload (requests/min)]



(d) MOEA/D-STM and read-only pattern [the axis from left to right are Response Time (ms), Energy Consumption (watt) and Workload (requests/min)]

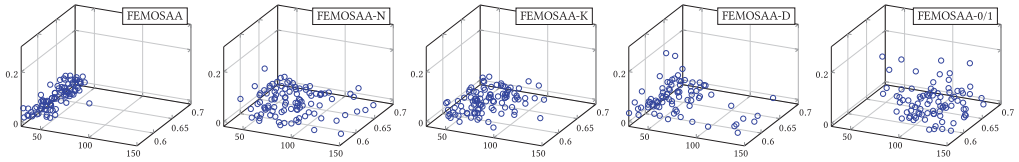


(e) NSGA-II and read-only pattern [the axis from left to right are Response Time (ms), Energy Consumption (watt) and Workload (requests/min)]

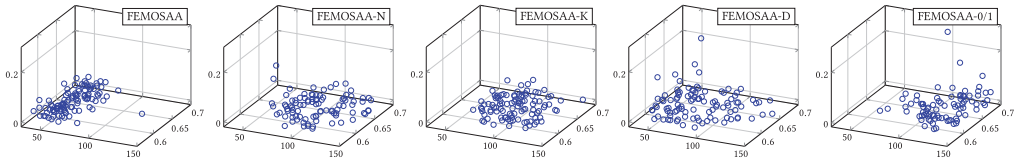


(f) IBEA and read-only pattern [the axis from left to right are Response Time (ms), Energy Consumption (watt) and Workload (requests/min)]

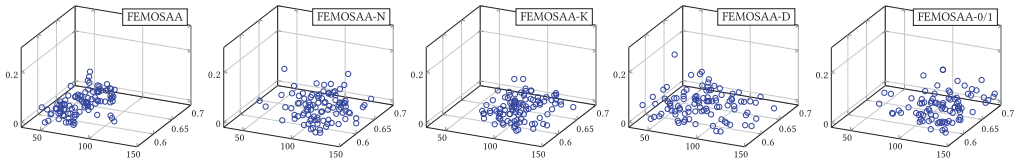
Fig. 30. Measured quality results with respect to workload for all the timesteps for RUBiS-SAS.



(a) MOEA/D-STM [the axis from left to right are **inverted** Throughput (request/second), Cost ($) and mean relative standard deviation of the candidate concrete services' quality]



(b) NSGA-II [the axis from left to right are **inverted** Throughput (request/second), Cost ($) and mean relative standard deviation of the candidate concrete services' quality]



(c) IBEA [the axis from left to right are **inverted** Throughput (request/second), Cost ($) and mean relative standard deviation of the candidate concrete services' quality]

Fig. 31. Measured quality results with respect to changes for all the timesteps for SOA-SAS.

effect sizes for all cases, as shown in Figures 27 to 29 and Figure 31. Those improvements of elitist chromosome representation over the conventional binary representation are mainly due to the fact that it fundamentally reduces the search space without affecting the original variability of SAS, so that the MOEA search has a larger chance to identify ideal solutions, leading to more valid solutions and better quality. In addition, as we show in Section 7.6, the elitist chromosome representation can significantly shorten MOEA running time.

Notably, as shown in Figure 30 for *RUBiS-SAS*, when the workload increases (i.e., more than 500 requests/min), we see that the achieved quality between FEMOSAA-N and FEMOSAA-0/1 barely differs. This is because the number of good and effective solutions tends to be limited when the workload is heavy, causing the benefit of reduced search space introduced by the elitist chromosome representation less obvious. Those results indicate that, overall, the elitist chromosome representation can better guide the search towards ideal solutions, but such improvement tends to blur as the workload increases. Similarly, for *SOA-SAS* in Figure 31, we see that although FEMOSAA-N creates better results than FEMOSAA-0/1 across different levels of diversity on the concrete services, the improvement tends to degrade under high diversity.

In summary, the results conclude that:

> **Answering *RQ1***—In contrast to the binary encoding, the elitist chromosome representation helps to produce better optimized quality for SAS with statistically significant results and nontrivial effect sizes on at least one quality attribute. It can also shorten the running time of MOEAs (in Section 7.6). However, the improvements might be hardly observed when the number of effected solutions is limited (e.g., under heavy workload).

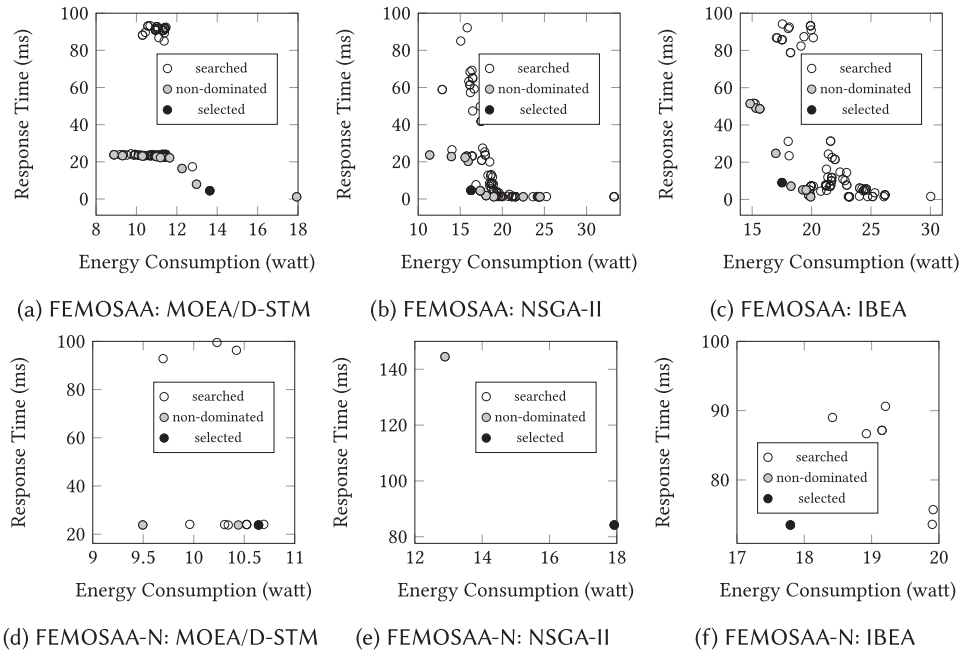


Fig. 32. Objective space of searched, nondominated, and selected valid solution(s) on a timestep of *RUBiS-SAS*.

7.4.2 Evaluating the Dependency-Aware Operators. To evaluate the benefit of dependency-aware operators, we first compare FEMOSAA with FEMOSAA-N for *RUBiS-SAS,* as shown in Figures 21 to 26. It is easy to see that, for all cases, FEMOSAA yields better results on both quality attributes with $p < 0.05$ and nontrivial effect sizes on at least one attribute. FEMOSAA also outperforms FEMOSAA-N in terms of HV, ED, and the number of valid solutions. In Figure 30, we see that the achieved quality of FEMOSAA better converges to the ideal area of the cube on different degrees of workload. Similar observations can be obtained for *SOA-SAS,* as illustrated in Figures 27 to 29 and Figure 31.

However, the preceding comparison does not indicate whether the improvement is mainly introduced by the dependency-aware operators or the knee selection method, thus, we then further compare FEMOSAA with FEMOSAA-K, which omitted the dependency-aware operators. In Figures 21 to 26 for *RUBiS-SAS*, we see that FEMOSAA still exhibits superior quality results under all cases which are statistically significant on at least one attribute with nontrivial effect sizes; it also comes with better HV and ED values, while the number of valid solutions is 100% versus less than 20%. In all cases, the achieved results of FEMOSAA, as shown in Figure 30, are more convergent around the ideal area of the cube, especially when the workload is heavy (i.e., more than around 500 requests/min). Similar results can be seen for *SOA-SAS,* as shown in Figures 27 to 29 and Figure 31. As one exception, we did observed that FEMOSAA-K is better than FEMOSAA on throughput when using MOEA/D-STM, the comparison is not statistically significant though. Such observation is an indication that the degree of objective conflicts in *SOA-SAS* is much stronger than in the case of *RUBiS-SAS*.

To gain a better understanding of why our dependency-aware operators can improve the quality of optimization, in Figure 32, we plot the objective space of all searched, nondominated, and selected valid solution(s) in the final population of an example run. We can clearly see that, under all studied MOEAs, the dependency-aware operators (i.e., FEMOSAA) can help to find solutions with

better convergence and diversity when compared with the cases where dependencies are ignored (i.e., FEMOSAA-N). Such a benefit eventually leads to a better set of nondominated solutions within which a final one can be selected for adaptation. Fundamentally, this is because the dependency-aware operators can better guide the search to prevent MOEAs from exploring invalid solutions, which virtually reduces the search space and provides a larger chance to find better solutions.

Interestingly, when we compare FEMOSAA-D with FEMOSAA-N in Figures 21 to 26 for *RUBiS-SAS*, we did not observe significant differences between them in terms of all the metrics, and their comparison has also failed in all statistical significance tests, which differs from our expectation that FEMOSAA-D should achieve statistically better results. The same can be also registered in Figure 30. We believe this is because although FEMOSAA-D can guide the search for better solutions, the fact that a solution from the final nondominated solutions is randomly selected for adaptation has obscured the benefits of preventing the exploration of invalid solutions as such a solution might be highly imbalanced for the conflicting objectives. On the contrary, for *SOA-SAS* in Figures 27 to 29 and Figure 31, we see that FEMOSAA-D generally outperforms FEMOSAA-N with better HV and ED values. In particular, under NSGA-II and IBEA, FEMOSAA-D is better on both attributes with $p < 0.05$ and nontrivial effect sizes. This implies that, for a feature model with relatively more complex dependencies as in the case of *SOA-SAS*, the dependency-aware operators can guide the search process to evolve many highly effective solutions in the final nondominated set; thus, the randomly selected adaptation solution, although imbalanced, could still be much better than those cases when dependencies are omitted.

In conclusion, the results suggest that:

> **Answering *RQ2***—In contrast to the classic and widely used operators, the benefit of dependency-aware operators is that they help to find solutions with better convergence and diversity, leading to better optimized quality for an SAS and with statistically significant results and nontrivial effect sizes on at least one quality attribute. Such improvement can be more obvious when the number of effective solutions is limited (e.g., when the workload is heavy). However, applying dependency-aware operators without ensuring balance of the selected adaptation solution might obscure its effectiveness.

7.4.3 Evaluating the Knee Selection. We have already shown that the combination of dependency-aware operators and knee selection can lead to results that outperform the case where both are omitted. We now evaluate whether the knee selection method itself can introduce benefit for runtime optimization of SAS. From Figures 21 to 26 for *RUBiS-SAS*, when comparing FEMOSAA with FEMOSAA-D in which knee selecting has been omitted, we see that FEMOSAA achieves superior HV and ED as well as better results on both quality attributes. The comparisons are statistically significant with nontrivial effect sizes on at least one attribute for 5 out of 6 cases. In Figure 30, we can see that the results of FEMOSAA are more balanced (i.e., points converge more to the bottom-left), and such improvement is more obvious when the workload is heavy under all cases. For *SOA-SAS*, we observed similar outcomes overall, as illustrated in Figures 27 to 29 and Figure 31, except that FEMOSAA-D has better throughput than FEMOSAA under NSGA-II due to the strong conflicts of objectives in *SOA-SAS*. These results show that our knee selection method in FEMOSAA contributes to the overall quality of adaptation by automatically selecting the solution with a balanced tradeoff on the objectives (i.e., a good sense of compromise) to execute the adaptation. The results achieved by those knee solutions are naturally the most preferable when relative preferences between objectives are unknown or it is too difficult to quantify them (which is common for SAS).

For all the studied MOEAs in Figure 32 of *RUBiS-SAS*, we see that, when compared with randomly selecting an adaptation solution from the nondominated set (i.e., FEMOSAA-N),

incorporating knee selection can indeed select a more balanced solution for adaptation (i.e., FEMOSAA) as it is closer to the bulge near the bottom-left area of the objective space.

Nevertheless, unlike our expectation, FEMOSAA-K and FEMOSAA-N do not yield statistically significant differences, as shown in Figures 21 to 26 and Figure 30 for *RUBiS-SAS*. This is because the effectiveness of knee selection is fundamentally dependent on the quality of the searched valid solution; thus, as we showed in the previous section, when the search process wastes efforts to explore invalid solutions, the quality of the solutions in the final population might be compromised, which would negatively affect the benefit introduced by our knee selection method. For the same reason, for *SOA-SAS* in Figures 27 to 29 and Figure 31, we see that although FEMOSAA-K slightly outperforms FEMOSAA-N on both attributes under all cases, the majority of the differences (5 out of 6) are not significant statistically.

Overall, the results indicate that:

> **Answering *RQ3***—In contrast to the randomly selected nondominated adaptation solution, knee selection helps to find a more balanced solution for adaptation, leading to better optimized quality for SAS with statistically significant results and nontrivial effect sizes on at least one attribute. Such improvement can be more obvious when the number of effective solutions is limited (e.g., when the workload is heavy). However, applying knee selection without ensuring the quality of searched valid solution can obscure its effectiveness.

7.5 Comparing FEMOSAA with State-of-the-Art Frameworks

To further evaluate the effectiveness of FEMOSAA, we compare it with the following state-of-the-art search-based frameworks from the literature:

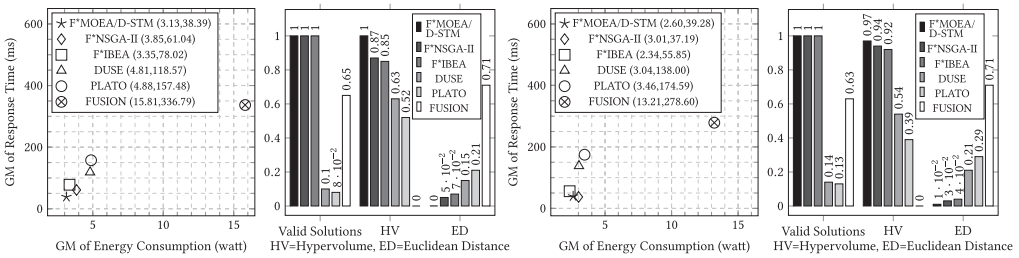
DUSE[10] [1]—A representative framework that optimizes SAS using NSGA-II. Since it does not consider dependency and knee in the optimization, we adapt the ad hoc strategies applied by FEMOSAA-D and FEMOSAA-K. As DUSE relies on manual transposition, we used elitist chromosome representations to ensure fair comparison and to eliminate bias in reproducibility.

PLATO [42]—An approach that applies a weighted sum of objectives and the Genetic Algorithm, a popular single-objective evolutionary algorithm, in the optimization. It also does not consider dependency in the optimization and thus we use the same ad hoc strategy as FEMOSAA-K. We specify equal weights on the objectives to find the single best solution. As PLATO relies on manual transposition, we used elitist chromosome representations to ensure fair comparison and to eliminate bias in reproducibility.

FUSION [23]—A well-known feature model-based framework that formulates the SAS optimization as an Integer Programming problem, which assumes aggregate objective function and is resolved by an exact algorithm (we use *Branch-and-Bound* in the experiments). FUSION uses binary representation of the solutions, and it considers categorical dependency only. Thus, when no valid solution found, we fix the violations on numeric dependency. To avoid unacceptable execution time, we forcibly return the best solution so far when it hits a time threshold (i.e., 40s).

Since the source codes of these state-of-the-art frameworks are not openly accessible, we have reproduced the implementation of the optimization algorithms that are core to these frameworks, following the exact guidance and setups mentioned in the work. Relevant open-sourced frameworks (e.g., jMetal [21]) are exploited when possible. In particular, we reproduced the representation of the solution and dependency handling with respect to these algorithms, as realized by the state-of-the-art frameworks. The other parts (e.g., the modeling component) are assumed to be

[10]DUSE is actually the same as FEMOSAA-N with NSGA-II which we have evaluated in the previous sections.



(a) Read-Write Pattern (b) Read-Only Pattern

Fig. 33. Comparing FEMOSAA (under different MOEAs) with state-of-the-art frameworks for RUBiS-SAS. (GM denotes Geometric Mean. F*MOEA/D-STM, F*NSGA-II, and F*IBEA denote FEMOSAA with MOEA/D-STM, NSGA-II, and IBEA, respectively.)

Table 2. Wilcoxon Signed-Rank Test Results Between FEMOSAA with Different MOEAs and State-of-the-Art Frameworks for RUBiS-SAS

(a) Read-Write Pattern

	p value (effect size) for Response Time (ms)			*p value (effect size) for Energy Consumption (watt)*		
	FEMOSAA with MOEA/D-STM	FEMOSAA with NSGA-II	FEMOSAA with IBEA	FEMOSAA with MOEA/D-STM	FEMOSAA with NSGA-II	FEMOSAA with IBEA
DUSE	**.004 (medium)**	**.043 (small)**	.327 (small)	**<.001 (large)**	**.032 (medium)**	**<.001 (large)**
PLATO	**<.001 (large)**	**.001 (large)**	**.015 (medium)**	**<.001 (large)**	**<.001 (large)**	**<.001 (large)**
FUSION	**<.001 (large)**	**<.001 (large)**	**<.001 (large)**	**<.001 (large)**	**<.001 (large)**	**<.001 (large)**

(b) Read-Only Pattern

	p value (effect size) for Response Time (ms)			*p value (effect size) for Energy Consumption (watt)*		
	FEMOSAA with MOEA/D-STM	FEMOSAA with NSGA-II	FEMOSAA with IBEA	FEMOSAA with MOEA/D-STM	FEMOSAA with NSGA-II	FEMOSAA with IBEA
DUSE	**.004 (medium)**	**<.001 (large)**	**.029 (medium)**	.246 (small)	.624 (trivial)	**.005 (medium)**
PLATO	**<.001 (large)**	**<.001 (large)**	**<.001 (large)**	**.010 (medium)**	.303 (small)	**.001 (large)**
FUSION	**<.001 (large)**	**<.001 (large)**	**<.001 (large)**	**<.001 (large)**	**<.001 (large)**	**<.001 (large)**

The significant statistics, i.e., $p < 0.05$, are highlighted and shown in bold.

identical to FEMOSAA in all experiments. We apply the same set of metrics, statistical tests, and methods for categorizing effect size as discussed earlier.

As shown in Figure 33, for both workload patterns on *RUBiS-SAS*, we can see that FEMOSAA with MOEA/D-STM, NSGA-II, and IBEA obtain better quality results than the other state-of-the-art frameworks. They also yield better results in terms of HV and ED. The statistical tests and effect size for the comparisons are shown in Table 2, in which we can see that the improvements obtained by FEMOSAA with all three MOEAs over the other state-of-the-art frameworks are statistically significant in general, resulting in $p < 0.05$ with nontrivial effect sizes on at least one quality attribute. The superiority of FEMOSAA when compared with DUSE again confirms that the combination of elitist representation, dependency-aware operators, and knee selection can guide the MOEA to achieve better quality results even when using different underlying MOEAs. PLATO, on the other hand, is further constrained by its nature of weighted sum objective functions, preventing it from finding some of the better tradeoff points. Although FUSION applies an exact optimization algorithm which should obtain the optimal solution, the fact of the given large search space has prevented it from reaching such an optimality in a reasonable time, thus forcing it to terminate when the time threshold is hit. Moreover, since the state-of-the-art frameworks do not consider all dependencies, their ratio of valid solutions ranges from 8.39% to 64.71% only. For the case of *SOA-SAS*, as shown in Figure 34 and Table 3, we see that FEMOSAA with all

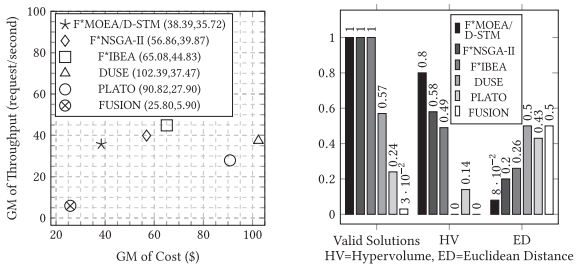


Fig. 34. Comparing FEMOSAA (under different MOEAs) with state-of-the-art frameworks for SOA-SAS. (GM denotes Geometric Mean. F*MOEA/D-STM, F*NSGA-II, and F*IBEA denote FEMOSAA with MOEA/D-STM, NSGA-II, and IBEA, respectively.)

Table 3. Wilcoxon Signed-Rank Test Results Between FEMOSAA with Different MOEAs and State-of-the-Art Frameworks for SOA-SAS

	p value (effect size) for Throughput (request/second)			*p value (effect size) for Cost ($)*		
	FEMOSAA with MOEA/D-STM	FEMOSAA with NSGA-II	FEMOSAA with IBEA	FEMOSAA with MOEA/D-STM	FEMOSAA with NSGA-II	FEMOSAA with IBEA
DUSE	.451 (trivial)	.300 (small)	**.007 (medium)**	**<.001 (large)**	**<.001 (large)**	**<.001 (large)**
PLATO	**<.001 (large)**	**<.001 (large)**	**<.001 (large)**	**<.001 (large)**	**<.001 (large)**	**<.001 (large)**
FUSION	**<.001 (large)**	**<.001 (large)**	**<.001 (large)**	**<.001 (large)**	**<.001 (large)**	**<.001 (large)**

The significant statistics, i.e., $p < 0.05$, are highlighted and shown in bold.

three MOEAs yields significantly better results ($p < 0.05$) than PLATO on both quality attributes. Through spending the highest cost, DUSE achieves higher value than FEMOSAA on throughput under MOEA/D-STM, but the comparison is statistically insignificant. Further, FEMOSAA is superior to and more balanced than DUSE overall, as evidenced by the greatly improved, statistically significant results on rest of the cases, as well as better HV and ED values. We can also observe that FUSION has very competitive results on cost, which is the best for this quality. However, this is due to the fact that the two quality attributes are strongly conflicting, which causes the exact search in FUSION to trap at a local area of the search space within the given time. This eventually resulted in an unwisely strong bias towards the improvement of cost, as evident by the even larger degradation on throughput in contrast to FEMOSAA, as well as the poor HV and ED values. FUSION leads to only 3% valid solutions as it considers categorical dependencies only while most of the cross-branched dependencies in *SOA-SAS* are numeric.

In conclusion, these results suggest that:

> **Answering *RQ4***—In contrast to the state-of-the-art frameworks (DUSE, PLATO, and FUSION), FEMOSAA achieves better optimized quality for SAS with statistically significant results and nontrivial effect sizes on at least one quality attribute, even when using different underlying MOEAs.

7.6 Runtime Overhead

Since we are interested in optimizing SAS at runtime, the running overhead of the MOEAs guided by FEMOSAA is an important aspect to evaluate. In Table 4, we compare the runtime overhead of FEMOSAA, FEMOSAA-N, FEMOSAA-0/1, DUSE, PLATO, and FUSION under *RUBiS-SAS*'s

Table 4. Comparing Mean Running Time for Producing an Adaptation Solution

	RUBiS-SAS with Read-Write Pattern	*RUBiS-SAS with Read-Only Pattern*	*SOA-SAS*
FEMOSAA with MOEA/D-STM	1.96s	1.88s	0.30s
FEMOSAA-N with MOEA/D-STM	2.04s	2.5s	0.43s
FEMOSAA-0/1 with MOEA/D-STM	22.02s	22.94s	1.06s
FEMOSAA with NSGA-II	0.9s	0.86s	0.08s
FEMOSAA-N with NSGA-II	0.94s	3.53s	0.04s
FEMOSAA-0/1 with NSGA-II	10.65s	10.48s	0.24s
FEMOSAA with IBEA	1.09s	1.12s	0.20s
FEMOSAA-N with IBEA	1.13s	1.12s	0.14s
FEMOSAA-0/1 with IBEA	20.90s	20.59s	1.45s
DUSE	0.94s	3.53s	0.04s
PLATO	0.99s	0.79s	0.02s
FUSION	37.98s	55.09s	54.54s

two different workload patterns and *SOA-SAS*. For FEMOSAA and its variants, we examine them using all the studied MOEAs. We report on the mean overhead over all the timesteps. As we can see, under all MOEAs and both subject SAS, FEMOSAA (less than 2s) yields much smaller overhead than FEMOSAA-0/1 (up to 22.94s) as the former encodes the elitist features only, which fundamentally reduces the search space and also simplifies the process of generating new solutions (individuals) in MOEAs. Interestingly, we note that for *RUBiS-SAS*, FEMOSAA (0.9s to 1.96s) has slightly smaller overhead than FEMOSAA-N (0.94s to 3.53s), which is quite surprising as we expected that the former should introduce slightly bigger overhead as it exploits additional processes in the reproduction operators and the selection of a final solution for adaptation. These results could be attributed to two reasons: (i) the extra efforts spent in dependency-aware operators and knee selection are negligible, and (ii) the dependency-aware operators tend to produce solutions that can affect the running time of MOEAs; for example, in NSGA-II, if the number of solutions in most of the higher ranked fronts is smaller, then the calculation of their crowding distances would yield less running time.

In contrast to the state-of-the-art frameworks, FEMOSAA achieves competitive results on runtime overhead, but the actual time taken by FEMOSAA depends on the underlying MOEA. Notably, FUSION has the biggest overhead as its exact algorithm fails to scale with a large search space and the optimization runs are often forcibly returned as they hit our predefined threshold of 40s.

In summary, the results reveal that:

> **Answering *RQ5***—While the running time of FEMOSAA depends on the underlying MOEA, FEMOSAA has very competitive runtime overhead in contrast to the state-of-the-art frameworks. In addition, the extra efforts spent in dependency-aware operators and knee selection are negligible; they may even slightly speed up the running time of MOEAs.

7.7 Discussion

7.7.1 FEMOSAA Benefits and Applicability. The most notable benefit of FEMOSAA is that it advances the synergy between software engineering for SAS and evolutionary computation. Without in-depth expertise on evolutionary algorithms in general, software engineers are granted the ability to influence the behaviors of a MOEA in whatever way they are familiar with (i.e., the feature

model design of a SAS). On the other hand, such a design serves as strong domain knowledge that can extend the MOEA and guide evolutionary search behavior (i.e., in form of elitist chromosomes and dependency-aware operators) to produce better solutions.

FEMOSAA exploits MOEA, which particularly fits for cases where the relative weights between objectives are unknown or it is too difficult to quantify them. From this perspective, the benefit of FEMOSAA is that it does not require one to specify weights, which could be labor-intensive. Indeed, we acknowledge that there are scenarios where the relative importance between conflicting objectives is explicitly known, and their weights can be precisely specified. In those cases, the concept of MOEA and knee selection employed by FEMOSAA might be less sensible. However, it is possible for FEMOSAA to work with single-objective evolutionary algorithms using aggregation of the objectives while deactivating knee selection, in which case the SAS can still benefit from the power of elitist chromosome representation and dependency-aware operators.

Another point worth mentioning is that MOEA does not guarantee optimal solutions; however, it is very efficient in producing good approximations to complex and nonlinear problems that would be otherwise unsolvable by exact optimization. Thus, we would not recommend using FEMOSAA on SAS that are simple, small in the search space, and can be handled by exact search that leads to an optimal solution.

7.7.2 FEMOSAA Running Time. Indeed, in contrast to FEMOSAA, the classic rule- and policy-based decision-making approach is very fast when adapting a SAS. However, the effectiveness of adaptation quality relies on several important factors, including:

(1) The full knowledge of every possible condition that the SAS may encounter.
(2) Some theoretical assumptions on the SAS and the environment that underpins the rules and policies.
(3) The manual reasoning of the optimal adaptation decision under a given condition (i.e., the mapping between a condition and an adaptation decision).

For SAS working in highly dynamic and uncertain environments (e.g., in cloud computing and software-defined networking), rule- and policy-based approaches would fail due to the fact that they heavily rely on human knowledge and there are emergent conditions that have not been accounted for, given the requirements of points (1) and (2). Further, SAS often has large variability (i.e., a large number of alternatives) as in the two SAS we studied. This makes point (3) in the rule- and policy-based approaches unrealistic; in fact, an exact search optimization would also fail under the problem of a large search space, as such a problem itself is intractable. In contrast, search-based software engineering techniques, particularly evolutionary algorithms, offer a promising way to solve our SAS optimization problem. This is because those algorithms are dynamic in nature and are able to perform optimization without in-depth knowledge and assumptions of the problem in hand (e.g., the property of the SAS and the environment). In addition, the notion of natural evolution and population permits it to find approximately optimal solutions even for intractable problems.

As we illustrated in Section 7.6, FEMOSAA achieves a runtime overhead of seconds under the two SAS studied. This may cause a delay in adaptation under extreme scenarios where the transition needs to be completed in microseconds. However, for other cases where the requirements of transition time can be relaxed, the better quality of adaptation generated by the underlying evolutionary algorithm has made the cost of its runtime overhead negligible, especially considering that the SAS optimization problem would otherwise be unsolvable using rule- and policy-based approaches. In fact, as we have shown throughout Section 7, the proposed elitist chromosome

representation and dependency-aware operators in FEMOSAA have enabled the underlying evolutionary algorithm to reach a better adaptation quality with even smaller runtime overheads.

7.7.3 Threats to Validity. Some threats to the validity of FEMOSAA follow.

Threats to construct validity are a concern when considering whether the used metrics can indeed reflect what we intend to measure. In this work, our experiments selected a wider range of quality attributes (i.e., response time, energy consumption, throughput, and cost). Those quality attributes and their metrics are the most commonly assessed quality aspects for SAS from the literature [39][49][23][42]. Further, we assessed the aggregated results of different quality objectives using HV and ED, which are widely applied metrics to measure the quality of solutions for multi-objective optimization problems [50][37]. Threats to construct validity could be also related to the stochastic nature of the considered MOEAs in experiments which can influence the measurements, especially for those that do not account for dependency and knee in the optimization. Indeed, to draw a meaningful conclusion of the measurements for stochastic search-based optimization, repeated runs are necessary, as suggested in Acuri and Briand [4]. We mitigated this bias by following the design introduced in Acuri and Briand [4], including conducting 102 optimization runs[11] for FEMOSAA and others, exploiting a statistical test to verify the significance of comparisons, and reporting the effect size to confirm that the measured results are not due to chance. Further, in the *RUBiS-SAS* case, FEMOSAA has been evaluated and measured following the realistic FIFA98 workload trace.

Threats to internal validity are related to the values of parameters for the MOEAs. The setup in this work has been carefully tailored to produce good tradeoff between the quality of optimization and the overhead. However, these values might vary depending on contextual characteristics (e.g., the given feature model, the types of SAS, and the environmental conditions), which itself could be a topic for future research. Threats to internal validity can also arise from accidental bugs in experimental implementations, which is always possible when writing any kind of software, especially for SAS, which is naturally highly complex. However, we tried to mitigate this unavoidable phenomenon by (i) exploiting well-established open-source frameworks whenever possible (e.g., jMetal [21]), (ii) following the exact guidance given in some of the compared work, and (iii) debugging through formal software testing procedure.

Threats to external validity are linked to the benchmark and scenario used in the experiments. To improve the generalization of experimental evaluations, we also evaluated FEMOSAA with three widely applied but distinct MOEAs under two running SAS and different workload patterns, which can diversify the runtime behaviors of the SAS. In particular, one of the SAS (i.e., *RUBiS-SAS*) contains a stack of real-world software that helps to emulate a more realistic scenario of the running SAS. Though our experimentation on the cases approximate real and industrial scale, it is difficult to claim complete generality; such a claim would require a much large number of independent domain-specific cases and needs to be performed by independent future adopters. In future work, we plan to evaluate FEMOSAA in other extreme contexts (e.g., in mobile environment where computational resources are rather limited). However, it is worth noting that the efforts in building the experiments and deploying the running SAS are nontrivial and could involve potential expenses.

8 RELATED WORK

Search-based optimization has been widely applied for SAS, either as a general framework or tailored to a specific domain (e.g., service or cloud systems). In this section, we provide an overview

[11] The number of runs is an implied result of the client emulator's setup for *RUBiS-SAS* and *SOA-SAS*.

of the most notable and relevant research in this area while examining them in the light of FEMOSAA.

8.1 Evolutionary Optimization for SAS

A common way of optimizing multiple objectives is to simply aggregate different objectives (e.g., weighted sum) such that they can be resolved in a single objective function. Hence, there exists some work that leverages single-objective evolutionary algorithms for optimizing SAS based on an objective aggregation. PLATO [42] and VALKYRIE [27] are two examples. However, as opposed to the automatic transposition approach in FEMOSAA, they rely on manual encoding of the SAS into chromosome representation, and they do not consider dependency between features. Further, it is well-known that the relative weights are difficult to tailor by engineers, and a single aggregation could restrict the search, causing limitations when searching for good solutions spread over the search space.

Exploiting MOEAs to handle the tradeoff between conflicting objectives is an emerging trend for optimizing SAS at runtime. Among others, DUSE [1] uses NSGA-II to produce a set of nondominated solutions for SAS. Similarly, NSGA-II and other EAs, as well as other meta-heuristic algorithms, have also been applied [28][22][51][12][15]; these focus on general SAS and the specific SAS for cloud and service systems. Nevertheless, their encoding of the SAS is manual, and no dependency between features is considered. Consequently, a significant number of function evaluations would be wasted in the search for invalid solutions, which, as we have shown, can degrade the quality of solutions found. Furthermore, these methods rely on the nondominated set, from which any solution can be used for adaptation, which can entail an imbalanced tradeoff. By contrast, FEMOSAA finds a knee solution that is generally preferable. Moreover, FEMOSAA relies on an automatic approach, where the elitist features are identified and the dependency is extracted to guide MOEA.

The closely related and most recent work is probably that produced by Pascual et al. [41], where they proposed a framework for a self-adaptive mobile system that uses the feature model with MOEA while considering feature dependencies in the optimization. However, FEMOSAA differs from their work in various aspects:

- Pascual et al.'s work relies on a binary representation of all features and only categorical features are considered, whereas FEMOSAA encodes elitist features into the chromosome only to form a polyadic representation that reduces the number of genes and thereby considerably shrinks the search space.
- Instead of modifying the reproduction operators, the work by Pascual et al. still allows the operators to explore and generate invalid solutions but fixes those solutions using a random repair strategy (i.e., each gene that violates the categorical dependency is fixed). However, such a fix is not guided by the dependency chain, thus there is no guarantee that the fixed gene would not cause additional violations (if there is a chain of dependency); therefore, it cannot ensure that a valid solution is always produced. Indeed, in their work, the process is repeated if the previous fix has not resolved all the violations, and the repair stops when a maximum number of repeats has been reached. On the contrary, FEMOSAA extracts dependencies and directly injects them to both the mutation and crossover operators, which are explicitly guided by the dependency chains and the related value trees; this fundamentally prevents invalid solutions from being explored.
- The nature of binary representation in Pascual et al.'s work implies that it is difficult for them to handle numeric dependencies, which are covered by FEMOSAA.

- FEMOSAA considers knee selection for adaptation, whereas Pascual et al.'s work selects any nondominated solution for adaptation, which could be highly imbalanced.

8.2 Evolutionary Optimization in Software Product Line Engineering

The feature model is widely used in Software Product Line (SPL) to model variability, which is similar to our use of a feature model on SAS. In particular, MOEA has been applied to SPL (see [45][46][31], for example). However, unlike SPL, where the objectives are highly concerned with designing software products that expose feature richness, diversity, and known defects, our focus in SAS is on optimizing nonfunctional quality attributes, seeking adaptation decisions that can better respond to dynamic situations and uncertainty in the environment with limited or no human intervention. In addition, SPL focuses on design time, while SAS development mainly focuses on runtime.

Even given the differences just mentioned, some SPL work exhibits resemblance to FEMOSAA: They both aim to transpose the design of the feature model onto the context of MOEA. Therefore, here, we compare FEMOSAA with those SPL preventative approaches where we specifically look at how the feature model is transposed into MOEA and the strategies used to handle feature dependencies. This consideration is particularly essential to explain why it is insufficient to directly apply the transposition used in existing studies of the SPL domain and to describe the improvements we have made in this aspect.

Among others, Sayyad et al. [45][46] exploit NSGA-II and IBEA for finding the optimal design of a product line using binary encoding of features into chromosome representations. In contrast, FEMOSAA does not rely on lengthy binary representation, and it encodes a polyadic chromosome using elitist features, which, as we have shown, leads to better optimization results and running time. Hierons et al. [31] proposed an extended MOEA encoding method for the feature model in which the basic encoding is still binary. Similar to our motivation, Hierons et al. [31] seek simplification by eliminating features that are the root of an $\mathcal{OR}$ group as their variability can be represented by their children (e.g., *Cache Mode* in Figure 4). However, FEMOSAA goes one step further by discarding the binary chromosome representation; this is achieved through selecting elitist features that cannot be removed without affecting variability while minimizing the length of encoding. This not only eliminates any features whose variability can be represented by its children (not restricted to those features that are the root of the $\mathcal{OR}$ group, as in Hierons et al. [31]; e.g., *Cache* and *Cache Mode* in Figure 4), but also eliminates those whose variability can be represented by their parent (e.g., *CPU*'s children in Figure 4). Unlike their work, we do not simply remove all mandatory features; we retain those with a $\mathcal{XOR}$ group of children as they would help us to considerably simplify the chromosome representation even more (e.g., *CPU* in Figure 4). Further, they have ignored numeric features, which are common in SAS. The numeric features, if not specifically handled, can lead to high computational overhead under binary encoding.

Another fact that distinguishes FEMOSAA from existing SPL work on the transposition process is the way in which the dependencies are handled. Hierons et al. [31] and Sayyad et al. [45][46] simply formulate dependency compliance as an additional objective to be optimized. Their formulation stems from the fact that it is too complex to explicitly handle dependency during evolution due to the inherited difficulty of the binary encoding. The basic assumption is that, with a large number of generations, the MOEA would eventually discover many valid solutions. While this schema may be sensible for design time optimization problems, it is ill-suited for SAS, where the optimizations occur at runtime, because the extra dimension of a dependency objective would impose too much additional difficulty on the problem and, at the same time, still fail to guarantee valid solutions. With the binary chromosome representation, Henard et al. [30] aim to overcome this by combining MOEA with an off-the-shelf CSP solver; here, the solvers act similarly to our

dependency-aware operators. However, these solvers are general and have not been specifically tailored to the needs of SAS problems; in addition, they are often computationally expensive and they would not guarantee valid solutions either. FEMOSAA, on the other hand, extracts the dependencies with respect to the elitist chromosome representation at design time, and those dependencies are injected into the operators of MOEA to fundamentally prevent the exploration of invalid solutions during runtime evolution. Further, FEMOSAA handles complex dependencies related to the numeric features.

8.3 Nonevolutionary Optimization for SAS

In addition to evolutionary algorithms, exact algorithms are also utilized for optimizing SAS [23][24][9] as they guarantee finding the optimal solution and can easily work with a single objective. Here, we consider the most noticeable nonevolutionary optimization approaches for SAS which do not tie to the special characteristics of a specific application domain. FUSION [23] is a general framework that optimizes SAS using exact algorithm. At design time, it also applies the feature model to represent the design of a SAS. However, unlike FEMOSAA, it does not consider numeric features, and it formulates the problem as an Integer Optimization problem using binary encoding of all features. Eshafani et al. [24] also follow an idea similar to FUSION, but they additionally use fuzzy logic to constitute the objective functions. MOSES [9] is a framework that is designed for self-adaptive service systems, where the optimization is formulated as a Linear Programming problem that is convex and can be solved exactly.

One fundamental issue with nonevolutionary and exact algorithms is that they fail to scale when the search space is large, which is often the case for modern SAS. Furthermore, exact approaches tend to be highly sensitive to the nature of the problem (e.g., whether they are convex or concave). This could impose extra difficulty because the analysis of the problem nature for SAS is very difficult, if not impossible, due to the dynamic and uncertain nature of the context. Additionally, exploiting exact algorithms often must work on an objective aggregation, which limits their applicability and capability.

In contrast, FEMOSAA relies on MOEA, which is a type of stochastic evolutionary optimization algorithm specifically designed to handle multi-objective optimization problems. It is known that MOEA can efficiently find optimal (or near-optimal) solutions for problems with a large search space, and it is capable of revealing a fine-grained tradeoff surface without the need for objective aggregation. In addition, MOEA is problem agnostic and therefore less sensitive to the nature of a given problem.

8.4 Other Approaches for Self-Adaptation at Runtime

Advanced control theory has also been used for SAS decision-making because of its low latency. Among others, Filieri et al. [25] propose a multi-objective controller where each objective would have independent sensors and actuators; the reasoning relies on aggregation of objectives, however. Simplex [47] is another recent control theory method for SAS, wherein a simplex optimization algorithm is used in conjunction with updates of controller gains.

Reinforcement Learning (RL) [8] is another thread that regards the SAS optimization problem as a learning problem. However, in RL, there is no explicit optimization process due to the absence of a clear model, and, therefore, the adaptation decision is often tailored in a trial-and-error manner that could impose an expensive exploration phase.

Without an explicit search behavior, both control theoretic and learning-based approaches lack the ability to perform exploration without affecting the SAS. In addition, they do not consider dependency constraints, and they are difficult to adapt for effective reasoning about tradeoffs at

runtime. In contrast, FEMOSAA exploits MOEA and the feature model, aiming to explicitly handle multi-objective optimization while considering dependencies.

9 CONCLUSION

This article presents FEMOSAA, a novel framework that systematically and automatically synergizes the feature model of SAS and a MOEA to optimize the SAS at runtime. At design time, FEMOSAA finds elitist features, including categorical and numeric ones, to create a polyadic chromosome representation; it then extracts dependencies between the genes, which are then used to extend the underlying MOEA for runtime optimization. The feature model serves as the engineers' domain knowledge that can reduce the search space (fundamentally and virtually) and guide the search, thus increasing the chance for finding better solutions. Further, FEMOSAA finds knee solutions that achieve a balanced tradeoff. By extensively comparing FEMOSAA with its variants and state-of-the-art frameworks on two complex real-world SAS, using three widely applied MOEAs and under two workload patterns for optimizing various conflicting objectives, we show that FEMOSAA produces statistically better and more balanced results for tradeoff with reasonable overhead. In particular, the most notable observations of FEMOSAA are that:

- Applying the elitist chromosome representation to encode the problem into MOEA helps to produce better quality and a smaller runtime overhead for optimizing SAS, but such improvement tends to become marginal when the number of effective solutions is small (e.g., the workload is heavy).
- The dependency-aware operators can properly guide the search, finding solutions with better convergence and diversity, leading to better quality SAS optimization. However, applying dependency-aware operators without ensuring a balance in the selected adaptation solution might obscure its effectiveness.
- The knee selection helps to find a more balanced solution for adaptation. However, applying knee selection without ensuring the quality of a searched valid solution can obscure its effectiveness.
- In contrast to the state-of-the-art frameworks from the literature, FEMOSAA, with all three studied MOEAs, produces statistically and practically better quality for optimizing SAS at runtime.
- Overall, FEMOSAA has very competitive runtime overhead in contrast to the state-of-the-art frameworks. Further, the extra efforts spent in dependency-aware operators and knee selection are negligible; occasionally, they can even slightly speed up MOEA runtime.

Our work impacts and advances the synergy between software engineering for SAS and evolutionary computation, combining strengths from both fields. Particularly with FEMOSAA, software engineers can exploit MOEAs to tackle SAS optimization without prior extensive expertise with MOEA. On the other hand, automatic transposition of the feature model into a MOEA context can improve MOEA and make the domain knowledge systematic and comprehensible for MOEA researchers, who in turn can design more effective algorithms for SAS. In contrast to many SBSE work, our deeper synergy takes one step further by automatically and dynamically extracting the domain information of SAS to extend the internal structure of MOEA. In future work, we plan to apply FEMOSAA in other domains of SAS and extend it to manage more conflicting objectives.

REFERENCES

[1] Sandro S. Andrade and Raimundo José de A. Macêdo. 2013. A search-based approach for architectural design of feedback control concerns in self-adaptive systems. In *Proceedings of the 2013 IEEE 7th International Conference on Self-Adaptive and Self-Organizing Systems*. IEEE, 61–70. DOI: http://dx.doi.org/10.1109/SASO.2013.42

[2] Apache Software Foundation. Apache Tomcat. Retrieved from http://tomcat.apache.org/. [Accessed 24 Mar 2018].
[3] Apache Software Foundation. Ehcache. Retrieved from http://www.ehcache.org/. [Accessed 24 Mar 2018].
[4] Andrea Arcuri and Lionel Briand. 2011. A practical guide for using statistical tests to assess randomized algorithms in software engineering. In *Proceedings of the 2011 IEEE 33rd International Conference on Software Engineering (ICSE)*. IEEE, 1–10.
[5] Martin Arlitt and Tai Jin. 2000. A workload characterization study of the 1998 World Cup web site. *Network Mag. of Global Internetworking* 14, 3 (May 2000), 30–37. DOI: http://dx.doi.org/10.1109/65.844498
[6] David Benavides, Sergio Segura, and Antonio Ruiz-Corts. 2010. Automated analysis of feature models 20 years later: A literature review. *Information Systems* 35, 6 (2010), 615–636. DOI: http://dx.doi.org/10.1016/j.is.2010.01.001
[7] Aurélien Bourdon, Adel Noureddine, Romain Rouvoy, and Lionel Seinturier. 2013. PowerAPI: A software library to monitor the energy consumed at the process-level. *ERCIM News* 92 (Jan. 2013), 43–44.
[8] Xiangping Bu, Jia Rao, and Cheng-Zhong Xu. 2013. Coordinated self-configuration of virtual machines and appliances using a model-free learning approach. *IEEE Transactions on Parallel and Distributed Systems* 24, 4 (2013), 681–690.
[9] Valeria Cardellini, Emiliano Casalicchio, Vincenzo Grassi, Stefano Iannucci, Francesco Lo Presti, and Raffaela Mirandola. 2012. MOSES: A framework for QoS driven runtime adaptation of service-oriented systems. *IEEE Transactions on Software Engineering* 38, 5 (Sept 2012), 1138–1159. DOI: http://dx.doi.org/10.1109/TSE.2011.68
[10] Tao Chen and Rami Bahsoon. 2013. Self-adaptive and sensitivity-aware QoS modeling for the cloud. In *Proceedings of the 8th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*. ACM/IEEE, 43–52.
[11] Tao Chen and Rami Bahsoon. 2017. Self-adaptive and online QoS modeling for cloud-based software services. *IEEE Transactions on Software Engineering* 43, 5 (2017), 453–475.
[12] Tao Chen and Rami Bahsoon. 2017. Self-adaptive trade-off decision making for autoscaling cloud-based services. *IEEE Transactions on Services Computing* 10, 4 (2017), 618–632.
[13] Tao Chen, Rami Bahsoon, Shuo Wang, and Xin Yao. 2018. To adapt or not to adapt? Technical debt and learning driven self-adaptation for managing runtime performance. In *Proceedings of the 9th ACM/SPEC International Conference on Performance Engineering (ICPE)*. ACM, 48–55.
[14] Tao Chen, Rami Bahsoon, and Xin Yao. 2014. Online QoS modeling in the cloud: A hybrid and adaptive multi-learners approach. In *Proceedings of the IEEE/ACM 7th International Conference on Utility and Cloud Computing*. 327–336.
[15] 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 20th International Genetic and Evolutionary Computation Conference (GECCO)*. ACM.
[16] Krzysztof Czarnecki, Simon Helsen, and Ulrich Eisenecker. 2004. Staged configuration using feature models. In *Proceedings of the International Conference on Software Product Lines*. Vol. 3154. Springer, 266–283.
[17] Rogério de Lemos, Holger Giese, Hausi A. Müller, Mary Shaw, Jesper Andersson, Marin Litoiu, Bradley Schmerl, Gabriel Tamura, Norha M. Villegas, Thomas Vogel, Danny Weyns, Luciano Baresi, Basil Becker, Nelly Bencomo, Yuriy Brun, Bojan Cukic, Ron Desmarais, Schahram Dustdar, Gregor Engels, Kurt Geihs, Karl M. Göschka, Alessandra Gorla, Vincenzo Grassi, Paola Inverardi, Gabor Karsai, Jeff Kramer, Antónia Lopes, Jeff Magee, Sam Malek, S. Mankovskii, R. Mirandola, J. Mylopoulos, O. Nierstrasz, M. Pezzé, C. Prehofer, W. Schäfer, R. Schlichting, D. B. Smith, J. P. Sousa, L. Tahvildari, K. Wong, and J. Wuttke. 2013. *Software Engineering for Self-Adaptive Systems: A Second Research Roadmap*. Springer Berlin, 1–32. DOI: http://dx.doi.org/10.1007/978-3-642-35813-5_1
[18] 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. DOI: http://dx.doi.org/10.1109/TEVC.2013.2281535
[19] Kalyanmoy Deb and Deb Kalyanmoy. 2001. *Multi-Objective Optimization Using Evolutionary Algorithms*. John Wiley & Sons, Inc., New York.
[20] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and T. A. M. T. Meyarivan. 2002. A fast and elitist multiobjective genetic algorithm: NSGA-II. *Transactions on Evolutionary Computing* 6, 2 (April 2002), 182–197.
[21] Juan J. Durillo and Antonio J. Nebro. 2011. jMetal: A java framework for multi-objective optimization. *Advances in Engineering Software* 42, 10 (2011), 760–771.
[22] Donia El Kateb, François Fouquet, Grégory Nain, Jorge Augusto Meira, Michel Ackerman, and Yves Le Traon. 2014. Generic cloud platform multi-objective optimization leveraging models@Run.Time. In *Proceedings of the 29th Annual ACM Symposium on Applied Computing (SAC'14)*. ACM, New York, 343–350. DOI: http://dx.doi.org/10.1145/2554850.2555044
[23] Naeem Esfahani, Ahmed Elkhodary, and Sam Malek. 2013. A learning-based framework for engineering feature-oriented self-adaptive software systems. *IEEE Transactions on Software Engineering* 39, 11 (Nov 2013), 1467–1493. DOI: http://dx.1452 doi.org/10.1109/TSE.2013.37
[24] Naeem Esfahani, Ehsan Kouroshfar, and Sam Malek. 2011. Taming uncertainty in self-adaptive software. In *Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering (ESEC/FSE'11)*. ACM, New York, 234–244. DOI: http://dx.doi.org/10.1145/2025113.2025147

[25] Antonio Filieri, Henry Hoffmann, and Martina Maggio. 2015. Automated multi-objective control for self-adaptive software design. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*. ACM, 13–24.

[26] Florian Fittkau, Sören Frey, and Wilhelm Hasselbring. 2012. CDOSim: Simulating cloud deployment options for software migration support. In *Proceedings of the 2012 IEEE 6th International Workshop on the Maintenance and Evolution of Service-Oriented and Cloud-Based Systems (MESOCA'12)*. IEEE, 37–46. DOI: http://dx.doi.org/10.1109/MESOCA.2012.6392599

[27] Erik M. Fredericks. 2016. Automatically hardening a self-adaptive system against uncertainty. In *Proceedings of the 11th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS'16)*. ACM, New York, 16–27. DOI: http://dx.doi.org/10.1145/2897053.2897059

[28] Sören Frey, Florian Fittkau, and Wilhelm Hasselbring. 2013. Search-based genetic optimization for deployment and reconfiguration of software in the cloud. In *Proceedings of the 2013 International Conference on Software Engineering (ICSE'13)*. IEEE Press, Piscataway, NJ, 512–521.

[29] Mark Harman, Edmund Burke, John Clark, and Xin Yao. 2012. Dynamic adaptive search based software engineering. In *Proceedings of the 2012 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*. ACM/IEEE, 1–8. DOI: http://dx.doi.org/10.1145/2372251.2372253

[30] Christopher Henard, Mike Papadakis, Mark Harman, and Yves Le Traon. 2015. Combining multi-objective search and constraint solving for configuring large software product lines. In *Proceedings of the 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering (ICSE)*. IEEE, 517–528.

[31] Robert M. Hierons, Miqing Li, Xiaohui Liu, Sergio Segura, and Wei Zheng. 2016. SIP: Optimal product selection from feature models using many-objective evolutionary optimization. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 25, 2 (2016), 17.

[32] Vigdis By Kampenes, Tore Dybå, Jo E. Hannay, and Dag I. K. Sjøberg. 2007. A systematic review of effect size in software engineering experiments. *Information and Software Technology* 49, 11–12 (2007), 1073–1086. DOI: http://dx.doi.org/10.1016/j.infsof.2007.02.015

[33] Kyo Kang, Sholom Cohen, James Hess, William Novak, and A. Peterson. 1990. *Feature-Oriented Domain Analysis (FODA) Feasibility Study*. Technical Report CMU/SEI-90-TR-021. Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA. Retrieved from http://resources.sei.cmu.edu/library/asset-view.cfm?AssetID=11231

[34] Joshua D. Knowles and David Corne. 2000. Approximating the nondominated front using the Pareto archived evolution strategy. *Evolutionary Computation* 8, 2 (2000), 149–172. DOI: http://dx.doi.org/10.1162/106365600568167

[35] Ke Li, Kalyanmoy Deb, Qingfu Zhang, and Sam Kwong. 2015. An evolutionary many-objective optimization algorithm based on dominance and decomposition. *IEEE Transactions on Evolutionary Computation* 19, 5 (2015), 694–716. DOI: http://dx.doi.org/10.1109/TEVC.2014.2373386

[36] Ke Li, Qingfu Zhang, Sam Kwong, Miqing Li, and Ran Wang. 2014. Stable matching-based selection in evolutionary multiobjective optimization. *IEEE Transactions on Evolutionary Computation* 18, 6 (2014), 909–923. DOI: http://dx.doi.org/10.1109/TEVC.2013.2293776

[37] 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 *Proceedings of the 40th International Conference on Software Engineering, NIER Track*. IEEE/ACM.

[38] Kaisa Miettinen. 1999. *Nonlinear Multiobjective Optimization*. Vol. 12. Kluwer Academic Publishers.

[39] Simon Mingay. 2007. Green IT: The new industry shock wave. *Gartner RAS Research Note G* 153703 (2007), 2007.

[40] Oracle Corporation. MySQL. Retrieved from https://www.mysql.com/. [Accessed 24 Mar 2018].

[41] Gustavo G. Pascual, Roberto E. Lopez-Herrejon, Mónica Pinto, Lidia Fuentes, and Alexander Egyed. 2015. Applying multiobjective evolutionary algorithms to dynamic software product lines for reconfiguring mobile applications. *Journal of Systems and Software* 103 (2015), 392–411. DOI: http://dx.doi.org/10.1016/j.jss.2014.12.041

[42] Andres J. Ramirez, David B. Knoester, Betty H. C. Cheng, and Philip K. McKinley. 2011. Plato: A genetic algorithm approach to run-time reconfiguration in autonomic computing systems. *Cluster Computing* 14, 3 (2011), 229–244. DOI: http://dx.doi.org/10.1007/s10586-010-0122-y

[43] Rice University. Rice University Bidding Systems. Retrieved from http://rubis.ow2.org/. [Accessed 24 Mar 2018].

[44] Nilabja Roy, Abhishek Dubey, Aniruddha Gokhale, and Larry Dowdy. 2011. A capacity planning process for performance assurance of component-based distributed systems. In *Proceedings of the 2nd ACM/SPEC International Conference on Performance Engineering (ICPE '11)*. ACM, New York, 259–270. DOI: http://dx.doi.org/10.1145/1958746.1958784

[45] Abdel Salam Sayyad, Joseph Ingram, Tim Menzies, and Hany Ammar. 2013. Scalable product line configuration: A straw to break the camel's back. In *Proceedings of the 2013 IEEE/ACM 28th International Conference on Automated Software Engineering (ASE)*. ACM/IEEE, 465–474.

[46] 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 *Proceedings of the 2013 International Conference on Software Engineering*. IEEE Press, 492–501.

[47] Stepan Shevtsov and Danny Weyns. 2016. Keep it SIMPLEX: Satisfying multiple goals with guarantees in control-based self-adaptive systems. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*. ACM, 229–241.

[48] University of Cambridge Computer Laboratory. Xen: A virtual machine monitor. Retrieved from http://www.xenproject.org/. [Accessed 24 Mar 2018].

[49] Hiroshi Wada, Junichi Suzuki, Yuji Yamano, and Katsuya Oba. 2012. E^3: A multiobjective optimization framework for SLA-aware service composition. *IEEE Transactions on Services Computing* 5, 3 (2012), 358–372.

[50] Shuai Wang, Shaukat Ali, Tao Yue, Yan Li, and Marius Liaaen. 2016. A practical guide to select quality indicators for assessing Pareto-based search algorithms in search-based software engineering. In *Proceedings of the 38th International Conference on Software Engineering (ICSE'16)*. ACM, New York, 631–642. DOI: http://dx.doi.org/10.1145/2884781.2884880

[51] Zeratul Izzah Mohd Yusoh and Maolin Tang. 2012. Composite SaaS placement and resource optimization in cloud computing using evolutionary algorithms. In *Proceedings of the 2012 IEEE 5th International Conference on Cloud Computing (CLOUD)*. IEEE, 590–597. DOI: http://dx.doi.org/10.1109/CLOUD.2012.61

[52] Qingfu Zhang and Hui Li. 2007. MOEA/D: A multiobjective evolutionary algorithm based on decomposition. *IEEE Transactions on Evolutionary Computation* 11, 6 (2007), 712–731. DOI: http://dx.doi.org/10.1109/TEVC.2007.892759

[53] Eckart Zitzler and Simon Künzli. 2004. Indicator-based selection in multiobjective search. In *Proceedings of the 8th International Conference on Parallel Problem Solving from Nature*. Springer, 832–842. DOI: http://dx.doi.org/10.1007/978-3-540-30217-9_84

[54] Eckart Zitzler, Marco Laumanns, and Lothar Thiele. 2002. SPEA2: Improving the strength Pareto evolutionary algorithm for multiobjective optimization. In *Evolutionary Methods for Design, Optimisation, and Control with Applications to Industrial Problems*. 95–100.

[55] Eckart Zitzler and Lothar Thiele. 1999. Multiobjective evolutionary algorithms: A comparative case study and the strength Pareto approach. *IEEE Transactions on Evolutionary Computation* 3, 4 (Nov 1999), 257–271. DOI: http://dx.doi.org/10.1109/4235.797969

Received July 2017; revised February 2018; accepted March 2018